VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2018 Jakub Štefanský

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Porovnání efektivity zpracování obrazu na různých HW platformách

Comparison of Effectiveness of Image Processing on Different HW Platforms

2018 **Jakub Štefanský**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Jakub Štefanský**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Porovnání efektivity zpracování obrazu na různých HW platformách
Comparison of Effectiveness of Image Processing on Different HW Platforms

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je porovnání typických úloh strojového vidění na dostupných HW platformách: PC, cRIO (Compact Reconfigurable Input Output), IC (Industrial Controller). CRIO je postaven na čipu ZYNQ, který obsahuje procesor ARM a programovatelné hradlové pole (FPGA). IC je průmyslový kontrolér postavený na procesorech Intel i7 (i5) a obsahuje také FPGA. Obecně je vykonávání algoritmů na FPGA velmi rychlé a z podstaty paralelní, ale problémem je výměna obrazové informace mezi FPGA a procesorem. Výsledkem bude srovnání doby zpracování obrazových dat při použití variantních vyhodnocovacích algoritmů.

Body zadání:
1. Seznámení se s architekturou a specifiky cRIO a IC.
2. Seznámení se s vývojovým prostředím LabVIEW a nadstavbovými moduly pro programování cRIO a IC.
3. Seznámení se s typickými algoritmy používanými pro zpracování obrazu a souvisejícími knihovními funkcemi.
4. Návrh, implementace a vyhodnocení testů.
5. Ověření funkce a zhodnocení.

Seznam doporučené odborné literatury:

[1] VLACH, Jaroslav, Josef HAVLÍČEK a Martin VLACH. *Začínáme s LabVIEW*. 1. vyd. Ilustrace Viktorie Vlachová. Praha: BEN - technická literatura, 2008, 247 s. ISBN 978-80-7300-245-9.
[2] BRESS, Thomas J. *Effective labview programming*. 1st ed. Allendale: NTS Press, 2013, 701 s. ISBN 1934891088.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Petr Bilík, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Rád bych poděkoval všem, jejichž rady přispěly ke zpracování bakalářské práce. Zvláště pak děkuji panu doc. Ing. Petru Bilíkovi, Ph.D. za vedení mé bakalářské práce.

V Ostravě, dne 30.4. 2018

.....

podpis

Abstrakt

Hlavní úkolem této bakalářské práce je porovnání tří hardwarových platforem na základě efektivnosti zpracování obrazu. Efektivnost je zjišťována na délce času za zpracování snímku.

Dvě platformy myRIO a Industrial Controller využívají FPGA čip pro rychlé zpracování dat spolu s procesorem. Třetí platformou je počítač. Půjde tedy o porovnání klasického procesoru s FPGA čipem, přičemž v případě FPGA čipu se musí data dostat ze strany kde operuje procesor na stranu FPGA, což může být úzkým hrdlem celého zpracování.

Zvolený algoritmus pro zpracování obrazu je Cannyho hranový detektor, který je výpočetně dostatečně náročný pro zjištění výkonových nedostatků jednotlivých platforem. Zvolený algoritmus je teoreticky popsaný, z čehož je následně vytvořený návrh a implementace do testů.

Výsledkem práce je tabulka s časy za zpracování obrazu, kde je porovnání i s knihovní funkcí LabVIEW. Je také teoreticky vypočítaná časová režie na přenos dat na FPGA.

Klíčová slova

LabVIEW, FPGA, Cannyho hranový detektor, myRIO, Industrial Controller, zpracování obrazu

Abstrakt

The main task of this bachelor thesis is the comparison of three hardware platforms based on the efficiency of image processing. Efficiency is determined over the length of the processing time.

Two platforms myRIO and Industrial Controller use a FPGA chip for fast data processing together with a processor. The third platform is a computer. This is a comparison of a classical CPU with a FPGA chip, and in the case of an FPGA chip, the data have to go from the side where the processor runs on the FPGA side, which can be a narrow throat of the whole processing.

The chosen image processing algorithm is the Canny edge detector, which is computationally enough to identify the performance deficiencies of individual platforms. The chosen algorithm is theoretically described, which is followed by the design and implementation of the tests.

The result of the thesis is a table with image processing times, which compares also with the LabVIEW library function. There is also a theoretically calculated time overhead for FPGA data transfer.

Keywords

LabVIEW, FPGA, Canny Edge Detector, myRIO, Industrial Controller, image proccesing

Obsah

1 Úvod ... 12

2 Popis hardwaru ... 13

2.1 NI FlexRIO ... 13

2.2 NI myRIO ... 13

2.2.1 Základní parametry ... 13

2.3 NI Industrial Controller ... 15

2.3.1 Základní parametry ... 15

2.4 Notebook Dell Vostro ... 16

2.4.1 Základní parametry ... 16

2.5 FPGA čip ... 17

2.5.1 Porovnání použitých FPGA čipů ... 17

3 LabVIEW ... 18

3.1 Základní popis ... 18

3.2 Nadstavbové moduly pro myRIO a IC ... 18

3.2.1 LabVIEW FPGA modul ... 18

3.2.2 Knihovní funkce pro zpracování obrazu na FPGA ... 19

4 Teoretický popis použitých algoritmů ... 21

4.1 Převod barevného modelu ... 21

4.2 Filtrace ... 21

4.2.1 Vyhlazení - Gaussův filtr ... 22

4.2.2 Detekce hran - Sobelův operátor ... 22

4.3 Cannyho hranový detektor ... 23

5 Návrh testů ... 25

5.1 Přenos dat ... 25

5.2 Programování na FPGA ... 26

5.3 Návrh algoritmu pro konvoluční masku ... 27

5.3.1 Gaussův filtr ... 29

5.3.2 Sobelův operátor ... 29

5.3.3 Hledání lokálních maxim ... 30

5.3.4 Prahování s hysterezí ... 31
5.3.5 Výpočty s okrajovými pixely ... 31
6 Implementace testů ... 33
6.1 Testy na platformě myRIO a IC ... 33
6.1.1 RT aplikace ... 33
6.1.2 Implementace algoritmu na FPGA ... 35
6.1.3 Implemetace převodu z RGB do modelu ve stupních šedi ... 38
6.1.4 Implementace konvoluční masky ... 38
6.1.5 Implementace hledání lokálních maxim ... 40
6.1.6 Implementace prahování s hysterezí ... 41
6.2 Testy na PC ... 42
7 Vyhodnocení testů ... 44
8 Závěr ... 48
Použitá literatura ... 50

Seznam použitých symbolů a zkratek

FPGA	Field Programmable Gate Array
NI	National Instrument
USB	Universal Serial Bus
VI	Virtual Instrument
PC	Personal Computer
FIFO	First In - First Out
RT	Real – Time
IC	Industrial Controller
PCI	Peripheral Component Interconnect
SDR	Shrunk D Ribbon connector
SoC	System on a chip
RAM	Random Access Memory
LUT	Look-Up Table
CLB	Configurable Logic Block
DMA	Direct Memory Access
RGB	Red Green Blue
SCTL	Single-Cycle Timed Loop
FXP	Fixed-Point
SGL	Single
U32	Unsigned integer 32 bit
HLS	High Level Synthesis
HDL	Hardware Description Language

Seznam obrázků

Obrázek 1: PXI FPGA Modul a Camera Link Adapter modulu [3] [2] *13*
Obrázek 2: Architektura myRIO 1900 [12] *14*
Obrázek 3: Architektura Industrial Controleru IC-3173 [13] *16*
Obrázek 4: Cluster Pixel Bus *19*
Obrázek 5: Průběh Gaussiánu G(x)[7] *22*
Obrázek 6: Ukázka gradientu hrany [7] *23*
Obrázek 7: Blokové schéma přenosu dat *26*
Obrázek 8: Porovnání While smyčky a Timed smyčky [5] *26*
Obrázek 9: Blokové schéma Gaussova filtru *28*
Obrázek 10: Postup zápisu dat do paměti *28*
Obrázek 11: Ukázka úrovní úhlu gradientu *30*
Obrázek 12: Ukázka hledání lokálního maxima *30*
Obrázek 13: Ukázka interpolace *32*
Obrázek 14: Blokové schéma RT aplikace *33*
Obrázek 15: Konfigurace FIFO paměti *34*
Obrázek 16: Hlavní sekvence *35*
Obrázek 17: Blokové schéma smyčky 1 *36*
Obrázek 18: Blokové schéma smyčky 2 *37*
Obrázek 19: High Throughput funkce pro dělení *38*
Obrázek 20: Indexace paměti *38*
Obrázek 21: Paměťový blok pro čtení *39*
Obrázek 22: Interpolace bočních pixelů *39*
Obrázek 23: Implementace jednoho řádku Sobelova operátoru *40*
Obrázek 24:Kód pro hledání lokálních maxim a prahování *40*
Obrázek 25: Blokové schéma prahování s hysterezí *41*
Obrázek 26: Ukázka hlavní implementace Canny detektoru na PC *42*
Obrázek 27: FIFO paměť *43*
Obrázek 28: Graf času za přenos *45*
Obrázek 29: Graf času celkem *46*
Obrázek 30: Snímek před zpracováním *46*
Obrázek 31: Snímek ze detekce hran Sobelem *47*
Obrázek 32: Snímek z Cannyho hranového detektoru *47*

Seznam tabulek

Tabulka 1 Základní parametry NI myRIO 1900 [12] *14*
Tabulka 2 Základní parametry IC - 3173 [13] *15*
Tabulka 3 Základní parametry Dell Vostro 5568 *16*
Tabulka 4 Časy za zpracování Cannyho hranového detektoru *44*
Tabulka 5 Vypočítané časy za algoritmus na FPGA *44*
Tabulka 6 Časy za zpracování vyjádřené v procentech *45*

1 Úvod

Cílem této bakalářské práce je porovnat celkem tři různé hardwarové platformy. První dvě jsou platformy myRIO a Industrial Controler, které využívají FPGA čip pro rychlé zpracování dat spolu s procesorem. Každá z platforem má rozdílné výkonové parametry. Třetí platformou je počítač. Půjde tedy o porovnání klasického procesoru s FPGA čipem, přičemž v případě FPGA čipu se musí data dostat ze strany kde operuje procesor na stranu FPGA, což může být úzkým hrdlem celého zpracování.

Porovnání je prováděné na základě zpracování obrazu. Zvolený algoritmus pro zpracování obrazu je Cannyho hranového detektoru. Jedná se o detekci hran o tloušťce jeden pixel. Zpracování obsahuje několik kroků, což dělá tento algoritmus výpočetně náročný. Po implementaci algoritmu bude zjišťován čas za zpracování na jednotlivých platformách.

V první části jsou rozebrány jednotlivé platformy, jako jsou parametry a vnitřní architektura. Dále jsou popsány rozdílí mezi FPGA čipy, které jsou v zařízení použity.

Ve druhé části je základní popis použitého vývojového prostředí LabVIEW, ve kterém jsou naprogramovány veškeré navrhnuté algoritmy. A dále jsou popsány knihovní funkce pro zpracování obrazu, které se nacházejí v LabVIEW FPGA modulu.

Ve třetí části je teoretický popis použitého algoritmu. Tento algoritmus je rozdělen na popis převodu RGB modelu, filtrace obrazu Gaussovým filtrem, detekce hran Sobelovým operátorem. A nakonec samotný popis Cannyho hranového detektoru, kterému předchází předešlá úprava signálu a dále zpracovává obraz přes hledání lokálních maxim, a nakonec prahování s hysterezí.

Dále je uveden vlastní postup řešení problematiky, který začíná rozborem přenosu dat, a pak návrhem algoritmu popsaného v teoretické časti. Při návrhu se musí vycházet z možností FPGA čipu. Pokračuje se popisem implementace testů, jak na straně FPGA, tak i na PC.

Na závěr je provedeno zhodnocení dosažených výsledků v podobě časů za zpracování.

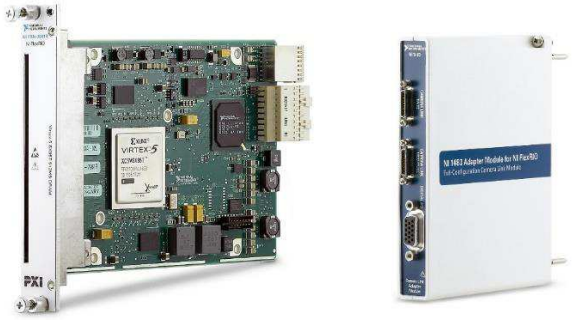
2 Popis hardwaru

2.1 NI FlexRIO

NI FlexRIO je hardware od společnosti National Instrument. Jeho hlavní předností oproti jinému hardwaru je, že data ze zdroje dat (vhodný HW) se posílají přímo na FPGA, kde se ideálně provede hlavní výpočetně náročné zpracování a pak mohou být posílána pro další zpracování do procesoru.

Základní částí je modul PXI FPGA, na které je samotný kód pro zpracování dat. FPGA může být programováno přes LabVIEW FPGA modul, nebo Verilog/VHDL. Modul PXI FPGA se připojuje do šasi buď přes PCI, nebo PCI Express. Mezi jednotlivými moduly v šasi může probíhat sdílení dat NI peer-to-peer, což nezatěžuje paměť vyhrazenou pro procesor.

Pro využití potenciálu na zpracování obrazu na NI FlexRIO existuje adaptér modulu Camera Link NI 1483. Podporuje standart Camera Link 1.2. Tento adaptér umožňuje pracovat ve čtyřech režimech base, medium, full and extend full. V režimu base stačí připojit kameru přes jeden 26 pinový SDR konektor, ve všech dalších režimech je potřeba použít dva konektory. V nejrychlejším režimu je schopen z kamery přenést 80 bitů za jednu periodu hodinového signálu, který má maximální frekvenci 85MHz. V tomto nejrychlejším režimu je tedy schopen přenést až 850 MB/s. Proto je použití NI FlexRIO s Camera Link adaptérem ideální pro kamery se snímkovací frekvencí řádově stovky za sekundu a také pro řádkové kamery s vysokým rozlišením. [1], [2], [3]



Obrázek 1: PXI FPGA Modul a Camera Link Adapter modulu [3] [2]

2.2 NI myRIO

2.2.1 Základní parametry

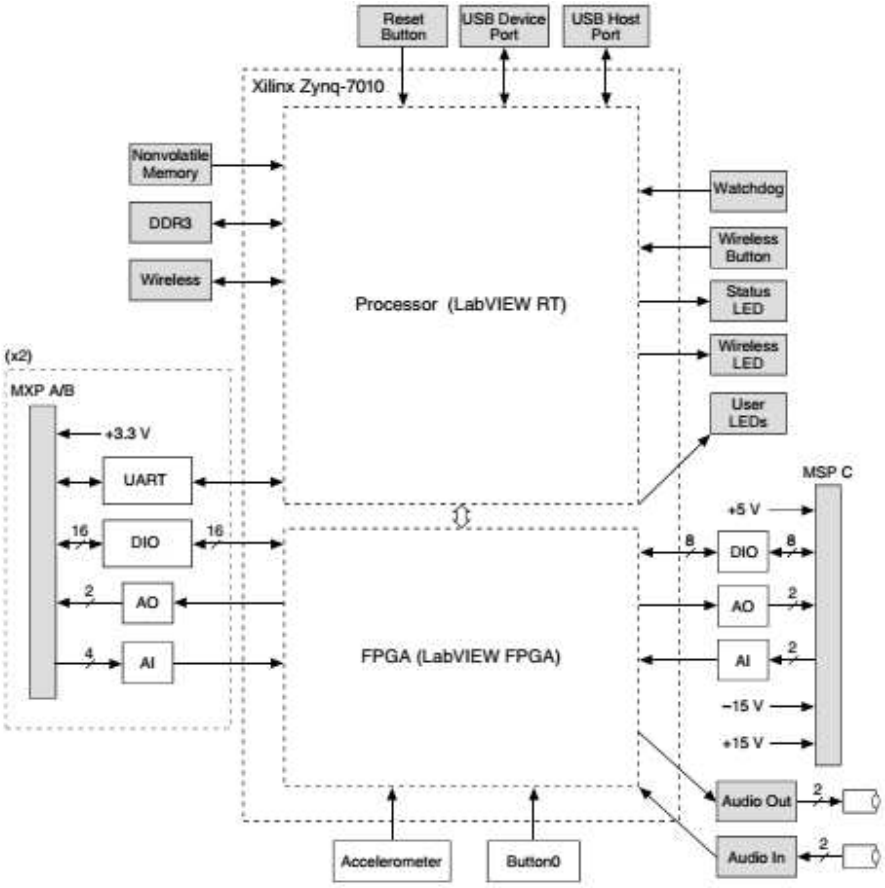
Jedná se o vestavěné zařízení společnosti National Instrument. V zadání této práce je využití zařízení compactRIO, které je v podstatě průmyslovým řešením zařízení myRIO a lze tedy po výkonové stránce nahradit. Zařízení myRIO je primárně určené pro akademickou oblast. [12]

Zařízení má rekonfigurovatelné vstupy a výstupy. Poskytuje analogové vstupy a výstupy, digitální vstupy a výstupy, zvukový a napájecí výstup. K hostitelskému počítači se připojuje přes USB rozhraní, nebo bezdrátově 802.11b. [12]

Základem NI myRIO je programovatelný systém na čipu Zynq-7010 SoC, který zahrnuje dvoujádrový procesor ARM Cortex-A9 a programovatelné hradlové pole Xilinx Artix-7. Na straně procesoru běží operační systém reálného času. [12]

Tabulka 1 Základní parametry NI myRIO 1900 [12]

Procesor	Dual-core ARM Cortex-A9 MPCore
Nonvolatilní paměť	512 MB
Operační paměť	256 MB 533MHz
FPGA čip	Xilinx Artix-7
USB port	2x USB 2.0 Hi-Speed



Obrázek 2: Architektura myRIO 1900 [12]

Na Obrázek 2 je ukázka vnitřní architektury čipu. Veškeré analogové a digitální vstupy a výstupy jsou přímo připojené na FPGA vstup. To umožňuje velmi krátkou odezvu na vstupní signál při zpracování na FPGA. Pro získání obrazu je, ale potřeba přístup k USB portu anebo k nonvolatilní paměti pro získání uložených snímků. A k těmto má přístup pouze RT procesor, který je propojený s FPGA. Proto je potřeba přenos dat z paměti RAM na FPGA, kde je využité tzv. AMBA AXI interconnect. Toto propojení je speciálně navržené pro nízkou latenci s krátkými cestami k paměti a dostatečnou šířkou pásma pro velký objem dat. Konkrétně je paměť RAM připojena na FPGA přes High-Performance AXI Porty. Porty můžou pracovat buď v 32 bitovém, nebo 64 bitovém šířce. [12], [14]

2.3 NI Industrial Controller

2.3.1 Základní parametry

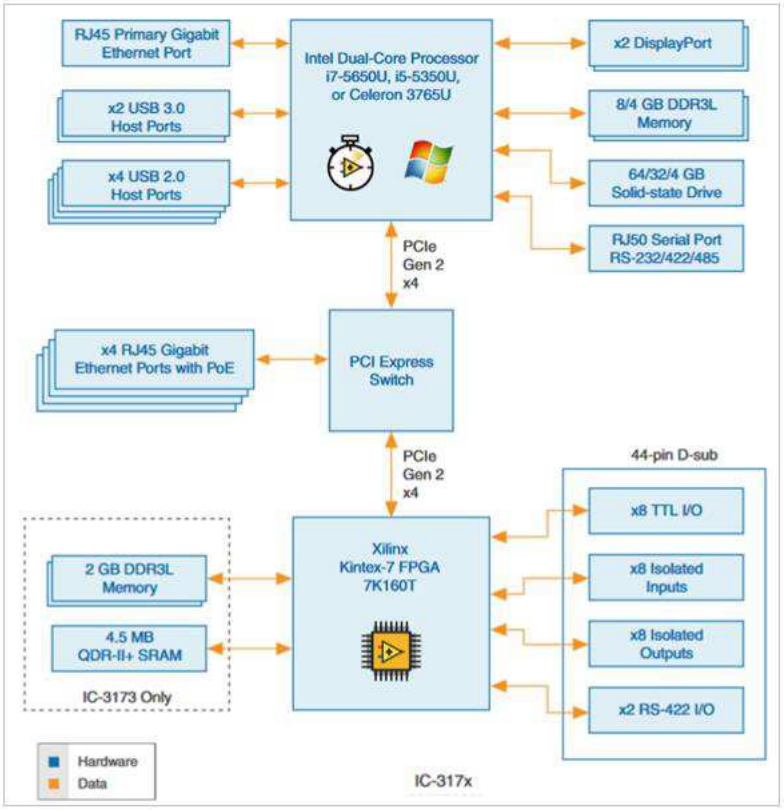
Jedná se o zařízení určené do průmyslu. Základem je dvoujádrový procesor Intel i7 páté generace. Procesor je taktován na frekvenci 2,2 GHz. Operační paměť má 8GB typu DDR3. Uložiště pro data má velikost 64 GB. Použitý programovatelný FPGA čip je Xilinx Kintex 7 160T. Na zařízení běží dvě verze operačního systému. První je Windows Embedded Standard 7 a druhý je operační systém reálného času NI Linux Real-Time. [13]

Tabulka 2 Základní parametry IC - 3173 [13]

Procesor	Dual-core Intel i7 2,2GHz
Nonvolatilní paměť	64 GB SSD
Operační paměť	8 GB
FPGA čip	Xilinx Kintex-7 XC7K160T
USB port	2x USB 3.0

Na Obrázku 3 je blokové schéma architektury IC. Výhoda této architektury tkví v kooperaci RT procesoru a FPGA čipu. FPGA čip v tomto případě je výhodný použít pro nejnáročnější zpracování s velmi rychlou odezvou na výstup na základě vstupu a procesor může potom zajistit vizualizaci výsledků a zpracování komplexnějších algoritmů. [13]

Obdobně jako u myRIO, jsou piny pro digitální vstupy a výstupy připojené na FPGA. Navíc je na FPGA připojena externí dynamická operační paměť DRAM a velmi rychlá statická operační paměť SRAM (označovaná jako cache). Propojení mezi procesorem a FPGA je řešené přes PCI Express Gen 2 x4. U Industrial Controlleru je USB port připojený k procesoru a také hlavní operační paměť a nonvolatilní paměť. [13]



Obrázek 3: Architektura Industrial Controleru IC-3173 [13]

2.4 Notebook Dell Vostro

2.4.1 Základní parametry

Jedná se notebook s dvou jádrovým procesorem Intel Core i5-7200U sedmé generace. Procesor je taktován na frekvenci 2,5GHz. Využívá operační paměť 8 GB DDR4 s frekvencí 2133MHz a úložný prostor má 256 GB s typem disku SSD. Disk je připojený přes slot M.2 SATA. Operační systém je Windows 10 Pro.

Tabulka 3 Základní parametry Dell Vostro 5568

Procesor	Intel Core i5-7200U
Nonvolatilní paměť	256 GB SSD
Operační paměť	8 GB
USB port	3x USB 3.0

2.5 FPGA čip

FPGA (Field-programmable gate arrays – Programovatelné hradlové pole) je programovatelný křemíkový čip. FPGA má stejnou flexibilitu jako běžící software na procesorovém systému, ale není omezen počtem procesorových jader. Na rozdíl od procesorů je zpracování už od přirozena paralelní. Proto každá nezávislá úloha na zpracování je vyhrazená určité části čipu a může tak pracovat samostatně bez vlivu ostatních logických bloků.
V důsledku není výkon jedné části aplikace ovlivněn při přidání dalšího zpracování. Zásadní výhodou je, že program je realizován hardwarem a neběží tedy v operačním systému. Tato skutečnost umožňuje velmi rychlou odezvu na výstup na základě vstupního signálu. [16]

Každý čip se skládá z konečného počtu předdefinovaných zdrojů s programovatelnými propojkami pro implementaci rekonfigurovatelného obvodu a I/O bloků, které umožnují přístup do vnějšího světa. FPGA zdroje zpravidla obsahují určitý počet konfigurovatelných logických bloků, logické bloky s pevně danou funkcí jako jsou bloky pro násobení, a vestavěné bloky RAM. [16]

Konfigurovatelné logické bloky (CLB) jsou základní logickou jednotkou FPGA. Někdy také označovány jako logické buňky obsahují nejčastěji klopné obvody (flip-flops) a vyhledávací tabulky(LUTs). Klopný obvod je využit jako posuvný registr, většinou pro uložení informace True nebo False do dalšího časového cyklu. A LUTs zajišťuje logické operace. LUT obsahuje pravdivostní tabulku s definovaným seznamem výstupů pro každou kombinaci vstupů (AND, OR, NAND a jiné). [16]

2.5.1 Porovnání použitých FPGA čipů

První FPGA čip je Xilinx Artix-7 obsahuje 28000 programovatelných logických buňek, 17600 vyhledávacích tabulek LUTs, 35200 klopných obvodů a 2,1Mb bloků RAM, kde jeden blok RAM má 36 Kb. [14]

Druhý čip Xilinx Kintex-7 XC7K160T využívá 162240 logických buňek, 25350 slices, kde jeden slice obsahuje čtyři LUTs a osm klopných obvodů. Dále obsahuje 325 bloků RAM po 36 Kb. [15]

3 LabVIEW

Jedná se o programovací a vývojové prostředí, kde zkratka LabVIEW znamená v předkladu "laboratorní pracoviště virtuálních přístrojů". Prostředí LabVIEW je produktem společnosti National Instrument. Tato platforma využívá tzv. G-jazyk (grafický jazyk), který je využíván pro vytváření měření a analýzu signálů, řízení a vizualizaci. Virtuální instrumentace umožňuje rychlý návrh nových aplikací a provádění změn v konfiguraci aplikace. [10]

V následujících podkapitolách bude základní popis prostředí, popis nadstavbových modulů pro zpracování obrazu na myRIO a IC.

3.1 Základní popis

Prostředí LabVIEW se skládá ze dvou základních částí, je to čelní panel a blokový diagram. Oba panely jsou provázané.

Čelní panel představuje uživatelské rozhraní vytvářené aplikace. Určuje vzhled a chování aplikace vzhledem k uživateli. Zobrazuje ovládací a indikační prvky, pomocí kterých lze ovládat běh a zobrazovat výsledky a stavy aplikace. [10]

Druhou částí je blokový diagram, ve kterém se implementuje potřebný algoritmus pomocí grafických prvků, které lze propojit datovými spoji. Tvar a barva datového spoje indikuje datový typ a typ objektu, například pole, nebo cluster a jiné. Na straně blokového digramu se automaticky zobrazí zvolené ovládací a indikační prvky ze strany čelního panelu. S těmito prvky pak lze pracovat propojením s dalšími prvky z palety knihovních funkcí. [10]

3.2 Nadstavbové moduly pro myRIO a IC

V této části budou popsány nadstavbové moduly týkající se zpracování obrazu na FPGA.

3.2.1 LabVIEW FPGA modul

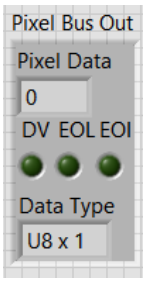
Na začátku vývoje FPGA čipů se začaly programovat pomocí tzv. hardware popisujících jazyků (HDLs) jako jsou VHDL a Verilog, které se vyvinuly do primárních jazyků pro programování FPGA čipů. Jedná se u nízko úrovňové jazyky. Z toho vyplývá, že pro programování FPGA je potřeba velká odborná znalost vnitřní architektury a funkce jednotlivých částí.

Proto došlo ke vzniku grafického nástroje pro návrh HLS, kterým je například LabVIEW FPGA modul. LabVIEW v tomto ohledu odstraňuje některé překážky tradiční návrhu aplikace ve HDL. Výhodou je, že grafické prostředí jednoznačně reprezentuje paralelismus a datový tok. Další je výhodou je, že nástroje kompilace LabVIEW FPGA automatizují proces kompilace. Pokud například dojde k chybě v časování v návrhu FPGA, tak LabVIEW zobrazí problémové části pro rychlejší odladění.

3.2.2 Knihovní funkce pro zpracování obrazu na FPGA

Společnost NI nabízí v LabVIEW FPGA modulu už hotové funkce pro zpracování obrazu na FPGA obvodu. Tyto funkce se dělí celkem do osmi kategorií. Jednotlivé funkce zde nejsou popsány, protože jsou poměrně obsáhlé, ale je popsán, alespoň základní přehled funkcí a jejich možností.

Základem jsou dvě funkce pro čtení a zápis dat z FIFO paměti. Přes tyto paměti je zajištěn přenos dat z Host na Target (v tomto případě FPGA) a zpět. Výstupem funkce je cluster (na Obrázek 4) zahrnující buď hodnotu jednoho, nebo osmi pixelů a další informace o datovém typu a také informace, jestli se jedná o pixel na konci řádku, nebo na konci obrazu. Tento výstupní cluster potom potřebuje na vstupu většina následujících knihovních funkcí pro zpracování obrazu. Cluster je pod označením Pixel Bus.



Obrázek 4: Cluster Pixel Bus

První kategorií jsou nástroje pro práci s barvou (Color Utilities), které se využívají pro práci jednotlivými složkami v obraze. Může to být extrakce konkrétní složky, převod hodnoty pixelu z jednoho barevného prostoru do druhého a jiné.

Druhou kategorií jsou funkce pro zpracování obrazu (Processing) ve stupních šedi anebo binárních. Použití je například pro transformaci jasu na základě definované vyhledávací tabulky, změna kontrastu v obraze, invertování hodnot pixelů a segmentace obrazu prahováním.

Třetí kategorií jsou funkce pro filtry (Filters). Lze použít funkce pro vyhlazení, odstranění šumu, nalezení hran, nebo i funkce pro zadání vlastní konvoluční masky.

Čtvrtou kategorií jsou funkce pro morfologické operace na obrazu (Morphology). Obsahuje funkce pro základní operace jako jsou dilatace a eroze na obrazu ve stupních šedi, nebo binárním.

Pátou kategorií jsou funkce pro základní zpracování na barevném obrazu (Color Processing). Například histogram, nebo prahování na barevném obrazu.

Šestou kategorií jsou funkce pro aritmetické a bitové operace (Operators). Funkce lze použít pro přičtení, odečtení, násobení a dělení obrazu s jiným obrazem, nebo konstantou. Další funkce umožní aplikovat logické operace a vytvoření porovnávaní pixelů mezi dvěma obrazy, nebo konstantou.

Sedmou kategorií jsou funkce, které vrátí informace o obraze (Analysis), který musí být ve stupních šedi, nebo binární. Funkce zjišťují histogram obrazu, dále informace o pixelech, anebo statistika podél jednodimenzionálního profilu v obraze.

Poslední kategorie obsahuje jednu funkci tzv. Posuvné měřítko (Caliper). Slouží pro detekci konkrétních okrajů v obraze.

V LabVIEW FPGA modulu je ještě možné využít expresní funkci Vision Assistent, která svým použitím připomíná Vision Builder, ale je omezena na funkce, které lze realizovat na FPGA. Funkce, které lze tedy využít ve Vision Assistentu jsou v podstatě funkce, které jsou popsané na předešlých řádcích, ale umožní jednodušší návrh.

V celé podkapitole je čerpáno ze zdroje [11].

4 Teoretický popis použitých algoritmů

V následujících podkapitolách jsou popsány algoritmy, které byly implementovány do testů v rámci bakalářské práce.

4.1 Převod barevného modelu

Základní barevný model je RGB. Kde R je červená, G je zelená a B je modrá složka. Každá barevná složka je zpravidla reprezentována 8 bity, potom výsledná barva vznikne složením v určitém poměru barevných složek. Tímto způsobem lze vyjádřit až 16,7 milionů barev ($2^{3\cdot 8} \cong 16{,}7 \cdot 10^6$). Někdy se používá přídavná složka A, která nese informaci o průhlednosti. [6]

Pro následující zpracování, v podobě detekce hran, je potřeba získat obraz z RGB modelu, ve kterém je získaný z kamery, na model ve stupních šedi. Kde úrovně šedi jsou vyjádřeny hodnotami jasu, které jsou vypočítaný na základě vzorce 4.1. Váhové koeficienty jsou definovány na základě citlivosti oka na různé barevné složky. Ze vzorce vyplývá, že největší citlivost je na zelenou složku, naopak nejmenší na modrou. [6]

$$Y = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \qquad (4.1)\ [6]$$

4.2 Filtrace

Filtrace se provádí za účelem snížení šumu, vyhlazení, zvýraznění a detekce hran. Konvoluce popisuje průchodu obrazu lineárním filtrem, což je základ filtrace obrazové funkce. Obraz je zpracován jako součin koeficientů masky s hodnotami vstupního obrazu s okolím O, tento postup se vykoná postupně pro každý pixel obrazu. Při filtraci se vyhází z předpokladu, že v daném okolí O mají pixely podobnou hodnotu jasu, proto lze upravit hodnotu středového pixelu na základě hodnot pixelů z okolí O. Lokální filtrace s nějakým okolím O, je vždy ztrátová. [7]

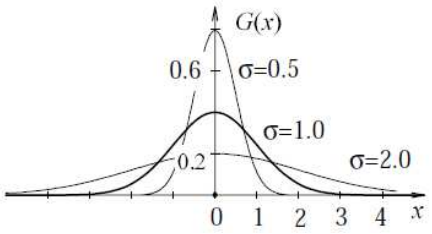
$$g_b(u,v) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s,t) \cdot f_b(u+s, v+t) \qquad (4.2)\ [7]$$

Vzorec 4.2 je matematický popis diskrétní konvoluce, která se používá pro aplikaci filtru. Výstupní obraz je $g_b(u,v)$. Konvoluční maskou je h s velikostí $m\ x\ n$, kde $m = 2 \cdot a + 1$ a $n = 2 \cdot b + 1,\ \ a, b \in N$. Rozměry masky jsou liché, aby mohl být počítaný pixel uprostřed masky. Nejčastěji je použitá čtvercová maska, kdy $a = b$. [7]

Základní dělení metod filtrace je na vyhlazení a detekce hran. Vyhlazení je zaměřené na potlačení šumu a osamocených špiček hodnot pixelů. Druhou metodou detekce hran je gradientní operátor, u kterého se provádí odhad první derivace z okolí O vstupního obrazu. Obě metody jdou proti sobě, to znamená, že čím větší vyhlazení tím je horší detekce hran vlivem rozmazání hran. [7]

4.2.1 **Vyhlazení - Gaussův filtr**

Tento typ filtru je tzv. Gaussovo vyhlazení, který používá konvoluční masku. Kde pixely blíže ke středovému mají vyšší váhu než krajní. Rozložení váhy vyplývá z Gaussovy křivky. Míra filtrace vychází z parametru σ, který udává strmost Gaussovy křivky. Na Obrázku 5 je ukázka Gaussovy křivky pro 1D signál. [6]



Obrázek 5: Průběh Gaussiánu G(x)[7]

Dvourozměrné Gaussovo rozdělení (vzorec 4.3) je definované od mínus do plus nekonečna, ale při implementaci diskrétní konvoluce je potřeba se omezit na nejčastěji využívané okolí 5x5 pixelů, kde středový pixel má pozici (0,0). Při výpočtu koeficientů masky se musí jednotlivé hodnoty normalizovat. Normalizované koeficienty musí po celkovém součtu dát hodnotu 1, proto je vždy potřeba výsledek podělit konstantou s velikostí celkového součtu koeficientů v matici. [6]

$$h(x,y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot exp(-\frac{x^2 + y^2}{2 \cdot \sigma^2}) \qquad (4.3)\ [6]$$

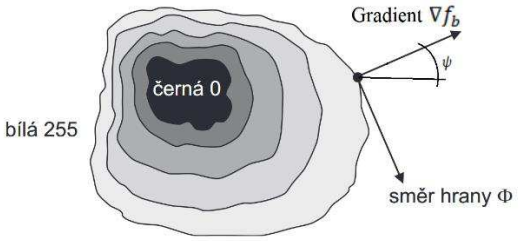
4.2.2 **Detekce hran - Sobelův operátor**

Hrana je definována, jako místo s náhlou změnou hodnoty jasu obrazové funkce $f_b(u,v)$. Pro nalezení těchto změn se využívá parciálních derivací a změnu funkce udává její gradient (vektorová veličina ∇f_b). Gradient určuje směr největšího růstu funkce (směr gradientu ψ) a strmost tohoto růstu (velikost gradientu $\|\nabla f_b(u,v)\|$). Směr je dán úhlem ψ mezi souřadnou osou u a rádiusem k bodu (u,v) v radiánech (vzorec 4.5). Vzorce platí pro spojitou obrazovou funkci, ale dají se s úpravami využít i u pro diskrétní obrazové funkce. Cílem praktického výpočtů je rozhodnout, zda pixel leží na hranici nějakého objektu. V jednoduchém případě lze považovat pixel za hranový, pokud velikost gradientu je nad nějakou prahovou hodnotou. Tento postup má ovšem nedostatky. První je, že hranice objektu vyjdou většinou tlustší jak jeden pixel. A hlavně tento postup neřeší problém doplnění chybějících a odstranění nadbytečných částí hranic. Tyto nedostatky řeší například Cannyho hranový detektor. [7] [8]

$$\|\nabla f_b(u,v)\| = \sqrt{\left(\frac{\partial f_b(u,v)}{\partial u}\right)^2 + \left(\frac{\partial f_b(u,v)}{\partial v}\right)^2} \qquad (4.4)\ [7]$$

$$\psi = arctan\left(\frac{\frac{\partial f_b(u,v)}{\partial u}}{\frac{\partial f_b(u,v)}{\partial v}}\right) \qquad (4.5)\ [7]$$

Pro hledání hran se využívají lokální gradientní operátory, které aproximují první parciální derivaci. Vychází se z předpokladu, že pixely s vysokou hodnotou gradientu jsou hranovými. Potom se tyto pixely spojují do hranic a směr hrany Φ je kolmý na směr gradientu. Ukázka je na obrázku 6. [7]



Obrázek 6: Ukázka gradientu hrany [7]

Gradientní operátory se využívají ve formě konvoluční masek, nejčastěji ve velikosti 3x3. Gradient je v takovém případě definován pro 8 směrů. Oproti filtrům s vyhlazením, musí součet koeficientů v masce gradientního operátoru dávat 0. V případě Cannyho detektoru je velmi často používaný Sobelův operátor. U kterého jsou využity dvě masky, jedna pro detekci vodorovných hran a druhá pro svislé hrany, ukázka je ve vzorci 4.6. Jelikož jednotlivé masky představují výpočet váženého průměru, měl by se výsledek podělit 1/4, ale pokud nezáleží na absolutní velikosti hrany, tak se může vypustit a kompenzovat následným vhodně zvoleným prahem při prahování. [7] [8]

$$h_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix},\ h_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad (4.6)\ [7]$$

4.3 Cannyho hranový detektor

V následující podkapitole je popsán Cannyho hranový detektor, který je popsán na základě vlastního pochopení tohoto algoritmu. Matematické odvození a důkazy jsou poměrně složité, proto v této podkapitole bude pouze slovní popis algoritmu, který je do určité míry upravený pro snazší návrh a následnou implementaci algoritmu. Základní struktura zůstává stejná.

Základní myšlenkou detektoru je, že skokovou hranu lze hledat filtrem. Detektor je optimální pro skokové hrany na základě třech kritérií:

1) Detekční kritérium – požaduje, aby významné hrany nebyly přehlédnuty a aby na jednu hranu nebyly vícenásobné odezvy.

2) Lokalizační kritérium – požaduje, aby rozdíl mezi skutečnou a nalezenou polohou hrany byl minimální.

3) Požadavek jedné odezvy – zajišťuje, aby detektor nereagoval na jednu hranu v obraze vícenásobně. Toto očekávání je již částečně zajištěno prvním kritériem. Tento požadavek je zaměřen zejména na zašuměné a nehladké hrany, což první požadavek nezajistí

Následující kritéria jsou doslovně citované ze zdroje [9].

Cannyho detektor má několik základních kroků. Nejdřív je potřeba hledat hrany ve známém směru. Obvykle není potřeba hledat ve všech směrech, ale postačí ve směru X a Y. Hledání se většinou provádí konvoluční maskou ve zvoleném směru například Sobelovým operátorem. Následuje výpočet velikosti gradientu a úhel gradientu, vzorce pro výpočet jsou v podkapitole 4.2.2. [8]

Když známe velikost a úhel gradientu pro každý pixel, tak se provádí tzv. hledání lokálních maxim. Tato operace se někdy nazývá potlačení odezev mimo maxima (non-maximal suppression). Vezměme například pixel, který leží na hraně. Pixel tedy bude představovat lokální maxim ve směru kolmém na hranu. Jak bylo napsáno v předešle podkapitole, tak na úhel gradientu je v kolmém směru hrana (Obrázek 6). Tyto vypočítané úhly je vhodné normalizovat ke zvoleným úrovním, proto aby nebylo potřeba interpolace hodnot pixelů ve směru úhlu a tím se zjednodušila implementace. [8] [9]

Finálně je potřeba provést prahování. Není vhodné použít prahování s jedním prahem, protože dochází často k rozpojování hran. Tento problém je účinně řešen prahováním s hysterezí, kde je stanoven horní a dolní práh. Pokud je pixel nad horním prahem, tak je uznán jako hranový, pokud je pod dolním prahem, tak je uznán jako pozadí. Pokud je pixel mezi prahy, tak je uznán jako hranový, pokud sousedí s již uznaným hranovým pixel, pokud nesousedí tak je nastaven na pozadí. Tímto způsobem dochází efektivně k eliminaci osamocených pixelů, které jsou mezi prahy. [8] [9]

5 Návrh testů

Pro zjištění efektivity zpracování obrazu je zvolený Cannyho hranový detektor, na kterém je zjišťována doba zpracování za jeden snímek. Toto zpracování je zvoleno z důvodu velké náročnosti na výpočetní výkon, ve kterém se můžou projevit nedostatky, nebo naopak výhody jednotlivých platforem. V případě platformy myRIO a IC, kde se obraz zpracovává na FPGA se do měřeného času započítá i čas nutný za přenos dat na FPGA a zpět. Rozlišení obrazu, na kterých se zjišťuje doba zpracování jsou zvoleny 320x240, 640x480 a 1280x720.

Při návrhu a následné implementaci zvoleného algoritmu není možné použít už předem vytvořené knihovní funkce v LabVIEW FPGA modulu, které jsou popsané v podkapitole 3.2.2. Důvodem je, že nelze nahlédnout do těchto funkcí (kód je uzamčen) a zjistit přesný algoritmus, který je použit, aby došlo k porovnání procesoru a FPGA. Proto je potřeba zvolený algoritmus sestavit ze základních prvků, aby bylo možné implementovat stejný algoritmus na FPGA a PC. Sestavení algoritmu se do značné míry odvíjí od možností a různých omezení přenosu a práce s daty na FPGA, které jsou popsány v jednotlivých podkapitolách návrhu.

5.1 Přenos dat

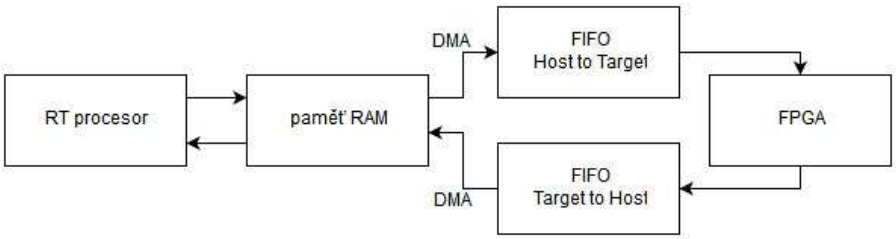
Přenos dat je potřeba řešit u platformy myRIO a IC, protože se u nich využije FPGA pro rychlejší zpracování kódu. Kamera se u obou platforem připojuje přes USB port, ke kterému má přístup RT procesor. Proto aplikace RT procesoru zajišťuje nastavení a snímání z kamery a následný přenos dat na FPGA.

Přenos mezi RT procesorem a FPGA je možný několika způsoby. Nejvhodnější pro přenos velkého objemu dat je metoda přímého přístupu k paměti DMA (Direct Memory Access). Tato metoda minimálně zatěžuje RT procesor. Kvůli rozdílnému času na zpracování se využívá paměť FIFO, ve které se hromadí data a podle toho jak rychle FPGA a RT procesor zpracovávají a čtou data z paměti. Nevýhodou DMA FIFO je režie volání spojená s každým přenosem, proto se hodí pro přenos velkého objemu dat najednou. Paměť pro DMA FIFO je přidělená jak na straně RT procesoru, tak na straně FPGA, ale chová se jako jedna FIFO paměť. [4]

Definované jsou dvě FIFO paměti. První paměť, musí mít velikost minimálně rovnu počtu pixelů v obraze a je nastavená na přenos Host to Target to znamená, že na straně RT procesoru se do paměti zapisuje a na FPGA se vyčítá jeden prvek za iteraci smyčky. Druhá paměť má stejnou velikost a je nastavena na přenos Target to Host. To znamená zápis hodnot na straně FPGA a na straně RT procesoru se budou data číst.

Při zápisu a čtení FIFO paměti na straně RT procesoru lze zapsat, nebo číst pouze 1D pole. Naopak na straně FPGA se čte a zapisuje vždy jeden prvek za iteraci smyčky.

Z Obrázku 7, kde je znázorněno blokové schéma přenosu dat, vyplývá, že data neprochází přes RT procesor, ale jsou hned přenášena z paměti RAM na FPGA na základě příkazu z RT procesoru. Tento postup zajišťuje dostatečnou rychlost pro přenos velkého objemu dat.

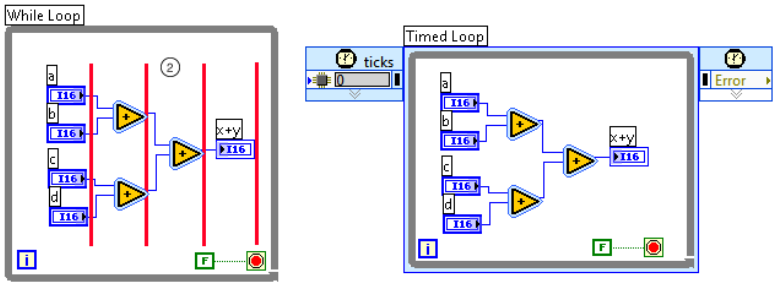


Obrázek 7: Blokové schéma přenosu dat

5.2 Programování na FPGA

Pro práci na FPGA je využitý LabVIEW FPGA modul. V tomto modulu je možné navrhovat program s dvěma přístupy zpracování kódu na FPGA, které jsou v této podkapitole popsány, protože od této volby se odvíjí návrh aplikace.

První možností je využít smyčku Single-Cycle Timed (dále jen SCTL) anebo druhou možností je While smyčka. V případě SCTL se jedná o časovanou smyčku, která se zpracovává podle zadané frekvence. Při použití této smyčky LabVIEW automaticky optimalizuje kód v této smyčce, aby se rychleji zpracovával a spotřeboval méně místa na FPGA. Zpracování je tedy rychlejší než v klasické While smyčce. Jak lze vidět na Obrázku 8, tak klasická While smyčka se provede ve čtyřech cyklech hodin a Timed smyčka se provede jenom v jednom cyklu hodin. Oba kódy se vykonají za jednu iteraci smyčky. Cyklus hodin musí být dostatečně dlouhý, aby se stihl kód provést. Pokud je čas příliš krátký, tak se neprovede kompilace. [5]



Obrázek 8: Porovnání While smyčky a Timed smyčky [5]

Jedním z omezení v SCTL smyčce je, že nelze v ní pracovat s proměnou s plovoucí desetinou čárkou, proto je potřeba použít datový typ s pevnou desetinou čárkou FXP. Zároveň FPGA obecně pracuje mnohem rychleji s tímto datovým typem, proto je vhodné použít tento datový typ i ve While smyčce. [5]

Dále není možné na FPGA využívat 2D pole, ale jenom 1D pole s předem přesně definovanou velikostí. Jelikož je pro přenos dat z procesoru na FPGA využitá DMA FIFO paměť, která je popsána v předešle podkapitole 5.1, tak hodnoty pixelů se vyčítají každou iteraci smyčky. Proto je potřeba jednotlivé hodnoty ukládat na FPGA do paměti pro další zpracování, protože například 2D konvoluce potřebuje nějaké zvolené okolí pro výpočet matice. LabVIEW FPGA

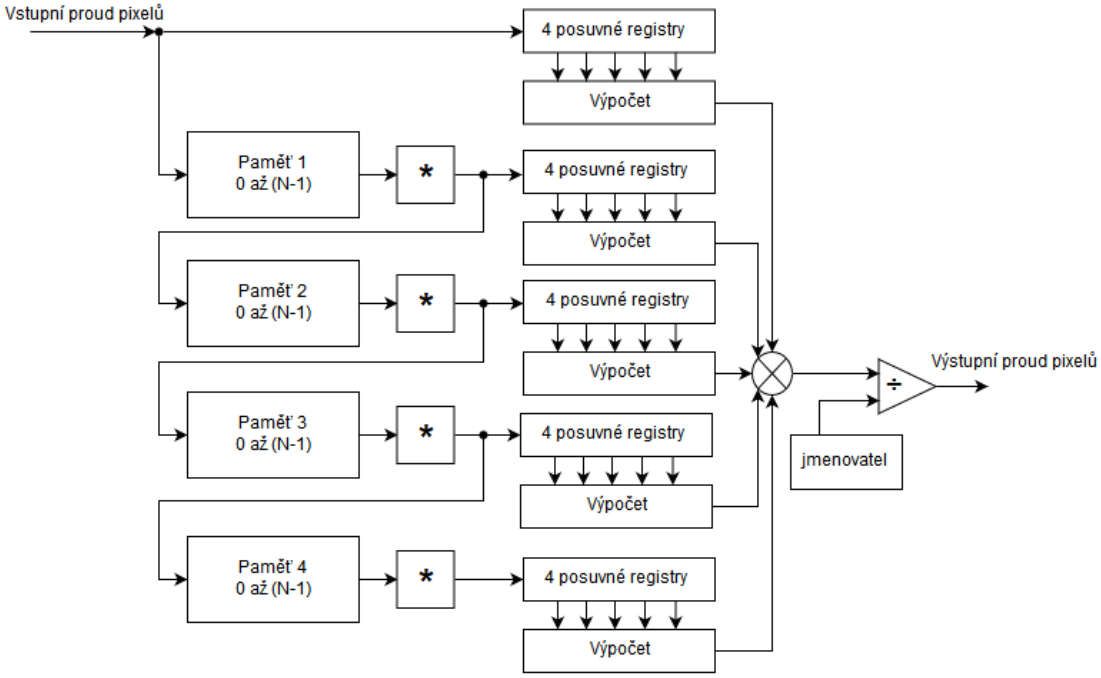
modul nabízí několik možností. První možností je Feedback Node. Jedná se o posuvný registr pro uložení několika hodnot mezi iteracemi ve smyčce. Další možností je pro uložení většího množství dat využití Memory Item. Tento prvek může pracovat v několika režimech. První možností je využití interní paměti RAM, někdy označovaná jako tzv. BRAM. K této paměti je možné přistupovat z SCTL smyčky, ale čtení z této paměti je možné se zpožděním minimálně jedenu iteraci, zápis je možný okamžitě v dané iteraci. Druhou možností je paměť LUT (Look Up table – Vyhledávací tabulka), která využívá samotných logických hradel na FPGA, proto je možné číst hned v dané iteraci bez zpoždění. Nevýhodou je, že spotřebovává logická hradla a tím snižuje velikost FPGA, který lze využít pro program. Proto se využívá pro malé množství dat. Ještě jedním typem paměti je externí DRAM, která ale není na použitém FPGA v myRIO a navíc není možné k ní přistupovat z SCTL smyčky. [11]

LabVIEW FPGA modul také nabízí funkce pro přenos dat mezi smyčkami. Například FIFO paměť, která má přednastavenou velikost. Tato funkce je hlavně vhodná pro přenos mezi smyčkami s různou taktovací frekvencí. Dále existuje funkce Registr, která ukládá jednu hodnotu a nehledě, jestli byla hodnota vyčtena dochází při dalším ukládání k přepsání předešlé hodnoty. Naproti tomu funkce HandShaking čeká na potvrzení o vyčtení hodnoty a až potom je možné přepsat tuto hodnotu na straně zápisu dat. [11]

5.3 Návrh algoritmu pro konvoluční masku

Popis návrhu je ukázaný na příkladu konvoluční masky s velikostí 5x5, která je využita u Gaussova filtru. Jak je už popsáno v podkapitole 5.1, tak z DMA FIFO paměti se každou iteraci vyčte jedna hodnota pixelu. Pro matici 5x5 je potřeba získat pět hodnot pixelů na řádek pro pět řádků v obraze. Kde pixel, pro který se počítá konvoluce je vždy uprostřed matice. Proto je potřeba hodnoty pixelů ukládat do pamětí, které jsou vytvořené pro každý řádek jedna.

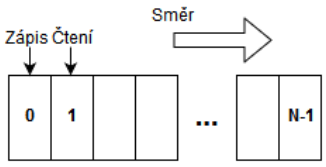
Jak vyplývá z Obrázku 9, kde je blokové schéma výpočtu Gaussova filtru, tak je použito čtyř pamětí, které mají velikost rovnu počtu pixelů na řádek obrazu. To znamená, že z každé paměti se vyčítat jedna hodnota, což dělá čtyři hodnoty na sloupec a pátá hodnota je právě vstupující do funkce. Potom je potřeba získat ještě pět hodnot na řádek, a to se docílí použitím čtyř posuvných registrů, které si uchovají čtyři hodnoty a pátá je na vstupu posuvných registrů. Tímto postupem vznikne matice 5x5 a je tedy možné provést vypočet konvoluce. Následuje sečtení jednotlivých hodnot a vydělení. Výstup zase představuje jednu hodnotu pixelu vztaženou k prostřednímu pixelu v matici.



Obrázek 9: Blokové schéma Gaussova filtru

Paměť pro zápis hodnot pixelů řádku musí fungovat na způsob kruhového bufferu. Postup zápisu a čtení je znázorněn na Obrázku 10, v podstatě se provede zápis na pozici, kde se v předešlé iteraci smyčky provádělo čtení. Provádí se, tak vyčtení jednoho řádku, kde během vyčítání se zapisuje nový řádek na pozice už vyčtených hodnot. Jak je popsáno v podkapitole 5.2, tak ideální pamětí pro zápis takového množství dat je interní paměť typu BRAM. Na Obrázku 9 je znázorněné blokem s hvězdičkou zpoždění jedenu iteraci při vyčtení hodnoty.

Pro to, aby byly výsledky vypočtených pixelů relevantní, musí být ve dvou pamětech načten celý řádek s hodnotami pixelů. A vzhledem k sekvenčnímu zápisu, to znamená, že se musí načíst nejdřív celá Paměť 1 a potom se začnou vyčítat a zapisovat správné hodnoty z Paměti 1 do Paměti 2. Po načtení dat do druhé se může začít interpolací hodnot pixelů pro zbylé dvě paměti a začít tak počítat relevantní hodnoty pixelů. Interpolace se provádí, dokud se nezapíšou správné hodnoty pixelů do dvou zbylých pamětí. Bližší popis interpolace okrajových pixelů je popsán v podkapitole 5.2.6.



Obrázek 10: Postup zápisu dat do paměti

Obdobný způsob aplikace konvoluční masky je použitý u výpočtu dvou konvolučních masek Sobelova operátoru, kde jsou použity jenom dvě paměti pro získání hodnot do matice 3x3. Pro hledání lokálních maxim a prahování se použije stejná část kódu jako u výpočtu Sobelova operátoru, přestože se nejedná konvoluci, ale využívá se okolí o velikosti 3x3.

5.3.1 **Gaussův filtr**

V této podkapitole je popsán volba konvoluční masky Gaussova filtru s velikostí 5x5. Zvolená maska Gaussova filtru 5.1 je vypočtená pomocí vzorce popsaného v teoretickém rozboru. Ze vzorce se vypočítají konstanty do matice, kde pro středovou hodnotu se volí pozice X=Y=0 a od této pozice se pro výpočet další hodnoty volí pozice o jedna větší. To znamená, že například pozice rohových konstant je X=Y=2. Výsledné hodnoty se zaokrouhlí pro zjednodušení aplikace konvoluční masky. Výsledná hodnota se ještě musí podělit, aby se získala výsledná hodnota 1. Proto se konstanty matice sečtou, v tomto případě 160 a touto hodnotou se podělí vypočtená hodnota z matice. Pro filtraci se konstanta rozptylu σ obvykle volí mezi 0,4 až 2 [8]. Jelikož při tomto návrhu není zásadní vytvořit nejideálnější filtr a volba parametru se liší podle velikosti šumu v obraze, proto je zvolen volitelně parametr filtrace σ = 1,4.

$$G(x,y) = \frac{1}{160} \cdot \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix} \qquad (5.1)$$

5.3.2 **Sobelův operátor**

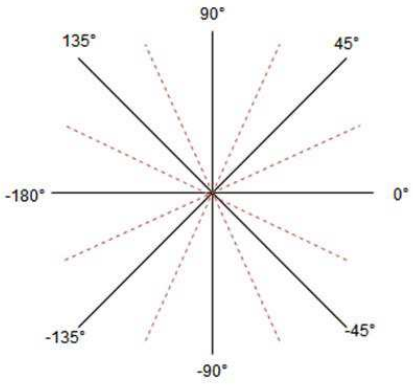
Detekce hran je provedena Sobelovým operátorem, který využívá konvoluční masku 3x3. Pro detekci hrany je využita jedna maska pro svislou hranu G_y a druhá maska pro vodorovnou hranu G_x .

$$G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix},\ G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad (5.2)\ [7]$$

Po výpočtu dvou konvolučních masek se získají hodnoty pro směr X a druhá pro směr Y. Jedná se o souřadnice gradientu. Z toho se provede výpočet velikosti toho gradientu přes Pythagorovu větu. A také se provede výpočet směru gradientu přes Arkus tangens. Obě hodnoty se využívají pro hledání lokálních maxim. Bližší teoretický popis Sobelova operátoru je v teoretické časti.

Vypočtený směr gradientu je v radiánech a pro další zpracování je potřeba přepočítat na stupně. Tento úhel ve stupních je nutné normalizovat ke čtyřem úrovním 0°, 45°,90° a 135°. Přičemž rozhodovací úroveň je poloviční úhel mezi dvěma normovanými úhly, například pro 0°a 45° je rozhodovací úroveň 22,5°. Záporný úhel se zrcadlově překlopí vzhledem ke směru,

například úhel -45° odpovídá po normalizaci úhlu 135°. Tímto postupem normalizace a překlopení záporných úhlů v daném směru se výrazně zjednoduší následná implementace, protože jak je popsáno v podkapitole hledání maxim, tak by se musely vzhledem k úhlům provádět interpolace mezi dvěma pixely, kterými by procházel daný směr gradientu středového pixelu. A po normalizaci bude směr gradientu zasahovat přímo sousední pixely bez nutnosti interpolace.



Obrázek 11: Ukázka úrovní úhlu gradientu

5.3.3 Hledání lokálních maxim

V této části algoritmu se provádí hledání lokálních maxim v okolí o velikosti 3x3.

Postup hledání maxima se provádí porovnáváním velikosti gradientu středového pixelu vzhledem k velikosti dvou sousedních pixelů. Zvolení sousedních pixelů se provádí na základě vypočteného úhlu gradientu středového pixelu, který je získaný ze Sobelova operátoru. Na Obrázku 12 je příklad, kdy středový pixel má vypočtený úhel gradientu 135° a proto se volí pixel1 a pixel8 pro porovnání. Je tedy potřeba vždy znát jenom úhel středového pixelu a není tedy potřeba hodnoty úhlu zapisovat do paměti.

Pokud středový pixel není největší než dva sousední, tak je uznán jako pixel pozadí, pokud naopak má nejvyšší velikost gradientu, tak se nemění a předává se pro další zpracování. Tato technika slouží pro ztenčení hran na pouhý jeden pixel.

pixel1	pixel2	pixel3
pixel4	$\|G_{xy}\|$	pixel5
pixel6	pixel7	pixel8

Obrázek 12: Ukázka hledání lokálního maxima

5.3.4 Prahování s hysterezí

V této podkapitole je popis prahování s dolním a horním prahem. Nejdřív je potřeba provést základní prahování, ideálně hned za hledáním lokálních maxim. Základní prahování se provede, tak že pixel je uznán jako hranový, pokud je nad horním prahem. A když je pod dolním prahem, tak je uznán jako pozadí. Pokud je pixel mezi dolním a horním prahem, tak bez změny pokračuje dál.

Další krok je samotné prahování s hysterezí. Zde se provede uznání pixelů za hranu, nebo za pozadí. Toto rozhodování se provádí pro pixely, které byly v předešlém kroku mezi prahy. Obdobně jako u předešlých částí algoritmu je potřeba matice 3x3. Potom se zjišťuje, jestli je v okolí středového pixelu matice 3x3 nějaký sousední pixel nad horní prahovou hodnotou, to znamená hranový pixel. Pokud je alespoň jeden sousední pixel hranový, tak je středový pixel uznán jako hranový. Tímto způsobem lze zabránit rozpojování hran a také eliminovat samostatné pixely způsobené šumem.

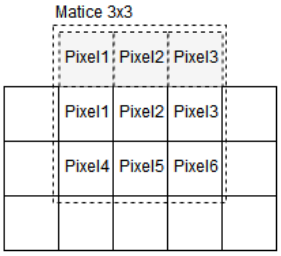
Pro správné prahování je potřeba ukládat výstupní hodnoty pixelů, protože jsou nutné pro rozhodování při zpracovávaní dalšího řádku. Tento postup zajistí kontinuitu hran. Ukládaný řádek obrazu nahrazuje třetí řádek ve zvoleném okolí 3x3.

5.3.5 Výpočty s okrajovými pixely

Při aplikování konvoluční masky, jak je znázorněno na Obrázku 13, je při použití matice 3x3 jeden řádek v matici mimo obraz. K okrajovým pixelům se může přistoupit, buď že se provede oříznutí výsledného obrazu o jeden pixel po celém obvodu obrazu, nebo se výpočet doplní nulami. Oba přístupy nejsou ideální, proto se v tomto řešení přistoupilo k interpolaci okrajových pixelů. Vychází z předpokladu, že při interpolací okrajů nedochází k velké chybě, protože při posunu o jeden pixel nebude příliš velká změna jasu. Provádí se tak, že se převezmou hodnoty pixelů z okraje do pomyslného nultého řádku nebo sloupce. Interpolace se tedy provádí také i na bočních stranách obrazu. Například v případě Gaussova filtru s maskou 5x5, je potřeba interpolovat dva řádky nahoře, dole a dva sloupce vlevo a vpravo. Pro snažíš implementaci je zvolena interpolace v místě výpočtu konvoluční masky a neprovádí se tedy interpolace celého řádku najednou.

Boční interpolace se provádí obdobně s tím, že se hodnota pro interpolaci bere ze středového pixelu, který leží na okraji obrazu. Bližší popis je popsaný v samotné implementaci.

Tomuto postupu interpolace se v odborné literatuře říká metoda nejbližšího souseda, kde se jedná o přiřazení hodnoty pixelu od nejbližšího souseda. [7]



Obrázek 13: Ukázka interpolace

6 Implementace testů

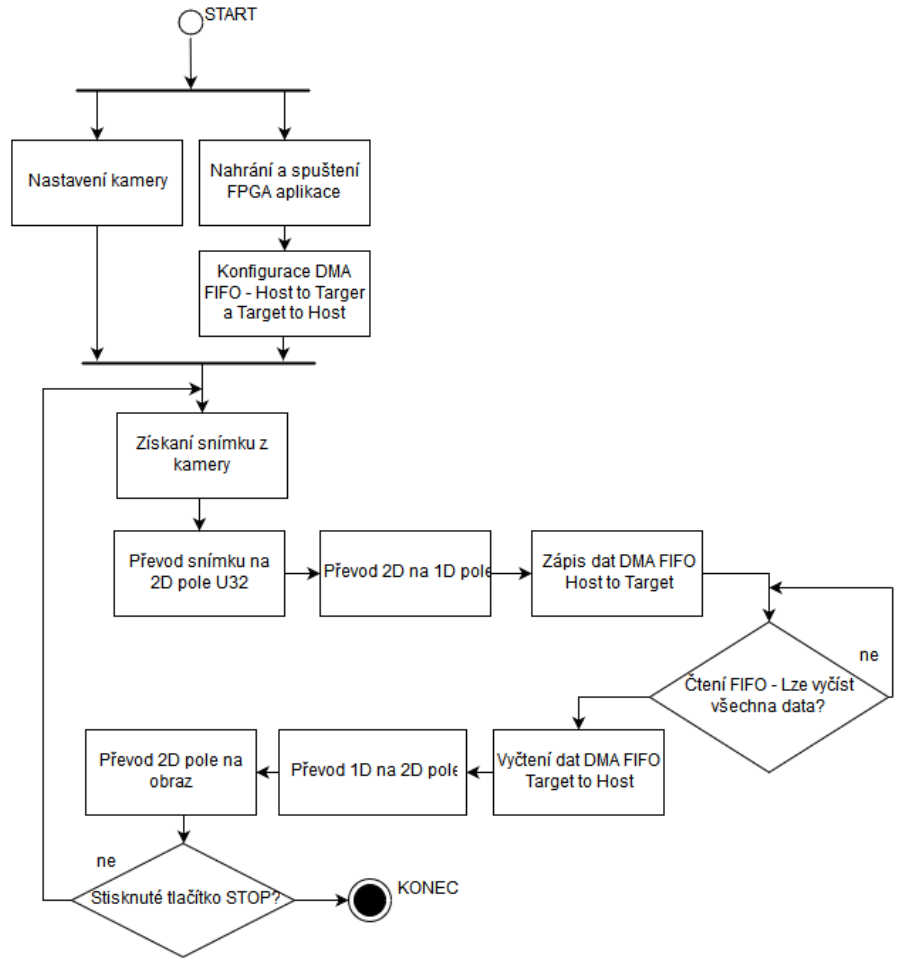
Zde jsou popsány jednotlivé realizace algoritmů na FPGA a PC v LabVIEW na základě provedeného návrhu.

6.1 Testy na platformě myRIO a IC

Na platformě myRIO a IC jsou dvě části SW, jeden program poběží na RT procesoru a druhý na FPGA. Na obou platformách poběží kompletně stejné programy, kromě drobných rozdílů v nastavení frekvencí a referencí.

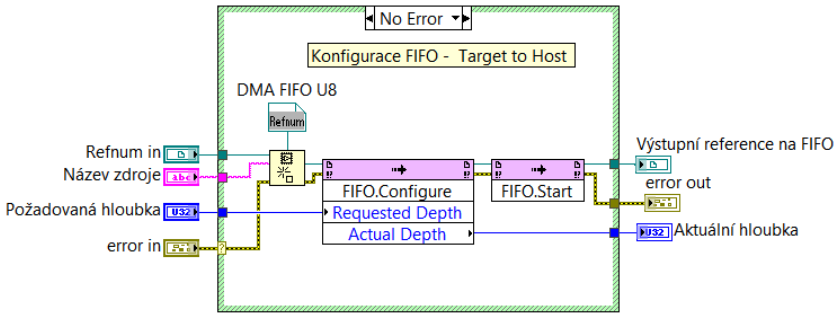
6.1.1 RT aplikace

Postup, jakým se vykonává kód RT aplikace je znázorněn v blokovém schéma na Obrázku 14.



Obrázek 14: Blokové schéma RT aplikace

Na začátku RT aplikace se provede nutná inicializace kamery pro kontinuální snímání, rozlišení obrazu apod. Dále se provede nahrání FPGA aplikace na čip a její spuštění. Na základě vzniklé reference na FPGA aplikaci se provede inicializace dvou pamětí. První je DMA FIFO nastavená pro přenos Host to Target, to znamená přenos ze strany RT procesoru na FPGA čip. A druhá paměť také DMA FIFO s typem přenosu Target to Host, která naopak přenáší data z FPGA čipu na RT procesor. Obě paměti mají velikost nastavenou na celkový počet pixelů zpracovávaného obrazu. Kód pro nastavení DMA FIFO je na Obrázku 15. Kde hloubkou paměti je myšlena velikost paměti.



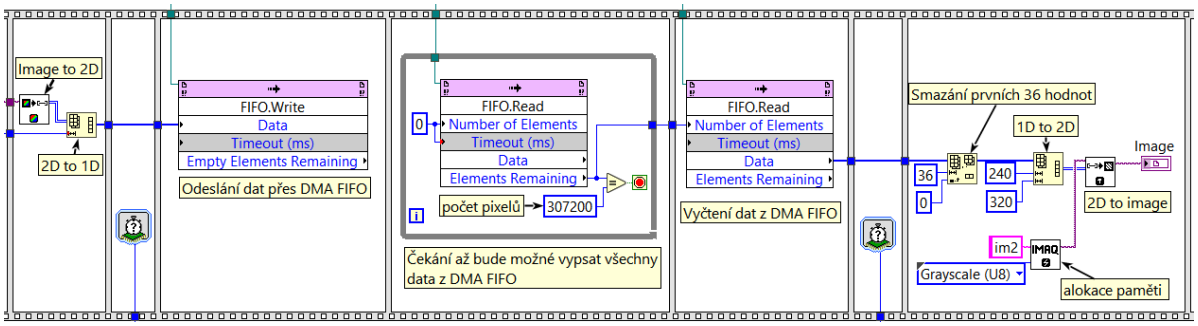
Obrázek 15: Konfigurace FIFO paměti

Po získání snímku se zahájí sekvence (Obrázku 16), která zajistí, že se jednotlivé dílčí úlohy se budou vykonávat sekvenčně. Sekvenčnost kódu je důležitá, aby nedošlo ke čtení dříve, než se zapíšou a zpracují data na FPGA a také umožní měření času jednotlivých úseků kódu.

V sekvenci se nejdřív snímek převede na 2D pole datového typu U32 (unsigned integer 32bi) pomocí funkce IMAQ ColorImageToArray. Jednotlivé byty této hodnoty U32 představují barevné složky RGBA, přičemž složka A se nevyužívá. Toto pole se musí transformovat na 1D pole funkcí Reshape Array, která se nastaví na celkový počet hodnot ve 2D poli. V dalším okně se 1D pole zapíše do FIFO paměti. Dále se spustí While smyčka, která běží, dokud se neprovede vyčtení, zpracování a zapsání dat na straně FPGA. Hodnota Elements Remaining přestavuje počet dostupných hodnot. Potom se mohou přečíst zapsané hodnoty ve FIFO paměti opět v podobě 1D pole. Potom se pole převede znovu na 2D pole a zobrazí se na panelu.

Ještě, než se 1D pole převede na 2D, tak se musí v poli odstranit prvních 36 hodnot, jedná se o chybné hodnoty vzniklé charakterem implementovaného algoritmu na FPGA. Chybné hodnoty vznikají použitými posuvnými registry v FPGA aplikaci. Jedná se o posuvné registry, které jsou použité pro uchování hodnot řádku, dále registry, které jsou součástí speciálních funkcí pro matematické operace High Throughput. Tyto posuvné registry mají na začátku chybné hodnoty do té doby, než se do nich provede zápis správných hodnot.

Hodnota horního a dolního prahu je pro FPGA aplikaci přenášené z RT aplikace. Používá se to přivedením reference na FPGA aplikace do metody Read/Write Control, která zajistí přenos hodnot do FPGA aplikace. Hodnoty prahu je tedy možné měnit za běhu aplikace.



Obrázek 16: Hlavní sekvence

Měření času se provádí za celou sekvenci a za přenos. Využívá se fukce TickCount, která se spustí v okamžiku, kdy přijde na řadu v sekvenci. Daná funkce je nastavená v režimu mikrosekund. Potom když dojde k odečtení dvou výstupů z funkce spuštěných v rozdílném čase, tak se získá celkový čas mezi tímto spuštěním. Smyčka je nastavena na celkem zpracování 100 hodnot, tímto způsobem vznikne pole časů, ze kterých se spočítá medián.

6.1.2 Implementace algoritmu na FPGA

Při implementaci Cannyho detektoru jsou využity dvě SCTL smyčky. V případě IC umožnila dostatečná velikost FPGA čipu taktovat obě smyčky na 90MHz. Ze vzorce 6.1 vyplývá perioda této smyčky 11,11ns. Naproti tomu smyčky na myRIO je možné taktovat jenom na 55MHz. Velikost frekvence se tedy odvíjí od náročnosti kódu a velikosti použitého čipu. V aplikaci na FPGA je navíc využit Feedback Node, který oddělí časově náročné části kódu, tím se provede paralelizace kódu, protože kód za Feedback Node nemusí čekat až dostane data, protože už má data z předešlé iterace. A to také umožní zvýšit taktovací frekvenci smyčky. Feedback Node odděluje jednotlivá subVI ve který se provádí zpracování obrazu v obou smyčkách.

$$T = \frac{1}{f} = \frac{1}{90000000} = 11{,}11ns \qquad (6.1)$$

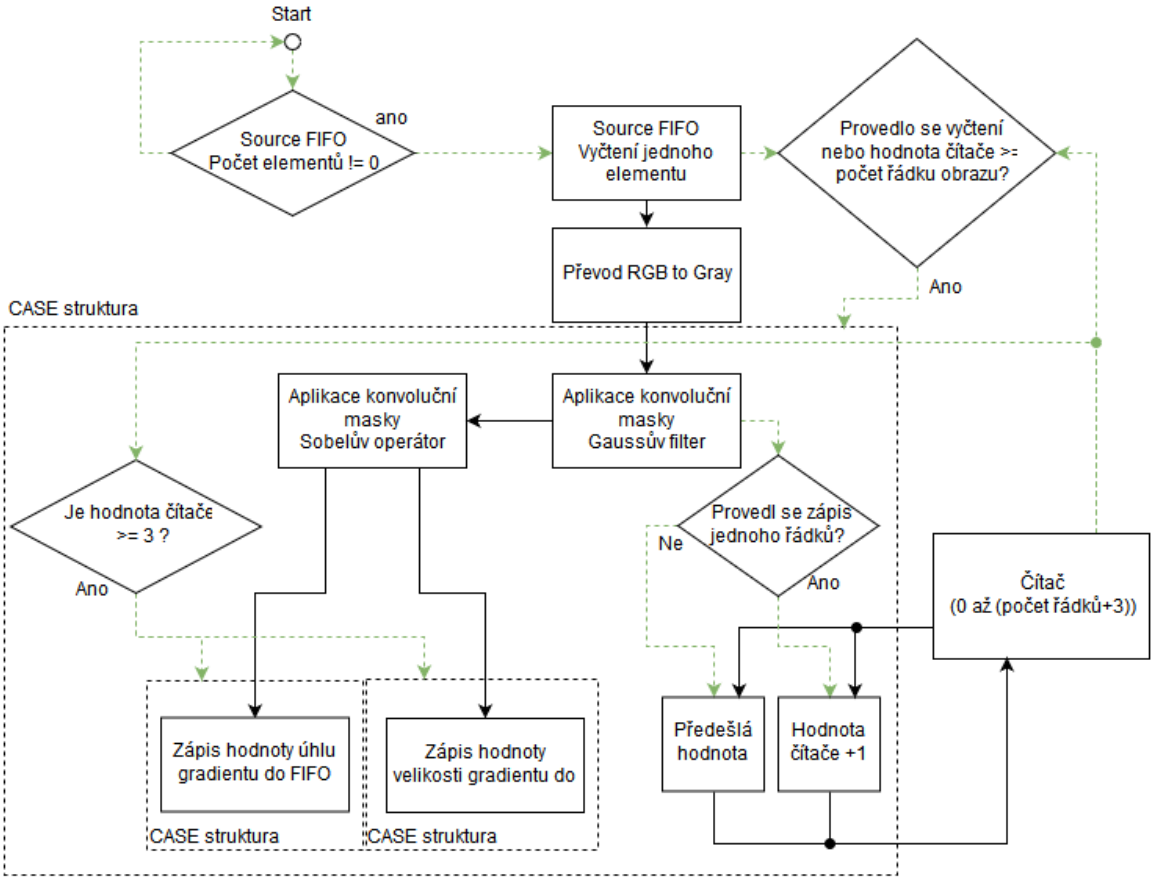
Smyčky zajišťují práci s vyčtením a zápisem dat a také obsahují subVI, což jsou vlastní vytvořené podprogramy pro zpracování obrazu. V této podkapitole je popis pouze funkce smyček, podmínek a spouštění jednotlivých subVI. Popis vnitřní funkcí subVI následuje v dalších podkapitolách.

Funkce první smyčky je znázorněná na Obrázku 17, která zajišťuje vypsání hodnoty z DMA FIFO paměti, ve schématu označená jako Source FIFO paměť. Z této paměti se vypíše prvek jenom pokud obsahuje nějaký uložený. Tato hodnota je převedena v subVI z RGB modelu na model ve stupních šedi. Potom následuje Case struktura, která je povolena, pokud se vyčetla hodnota, nebo je počet řádků už zapsaných hodnot větší, nebo roven počtu řádků obrazu. V samotné struktuře

se provádí subVI pro konvoluční masku Gaussova filtru, potom následuje subVI pro dvě konvoluční masky Sobelova operátoru pro směr X a Y. Výsledkem je hodnota velikosti gradientu a směr tohoto gradientu.

První smyčka ještě obsahuje čítač, který zvětší čítanou hodnotu o jedničku, pokud dojde k zápis jednoho řádku do paměti subVI Gaussova filtru. Jedná se tedy o čítač řádků už načtených do funkce. Bližší popis pamětí a ukládaní dat je v návrhu a implementaci filtru. Nicméně hodnota čítače se využívá pro povolení Case struktur pro ukládaní dat ze subVI Sobelova operátoru. K povolení dojde, pokud je hodnota čítače větší nebo rovno tři. Konstanta tři je dána tím, že musí být načtené tři řádky, aby byly výsledky z funkcí relevantní a mohli se zapisovat do výstupních FIFO pamětí. FIFO paměti jsou pro úhel gradientu a druhá pro velikost gradientu.

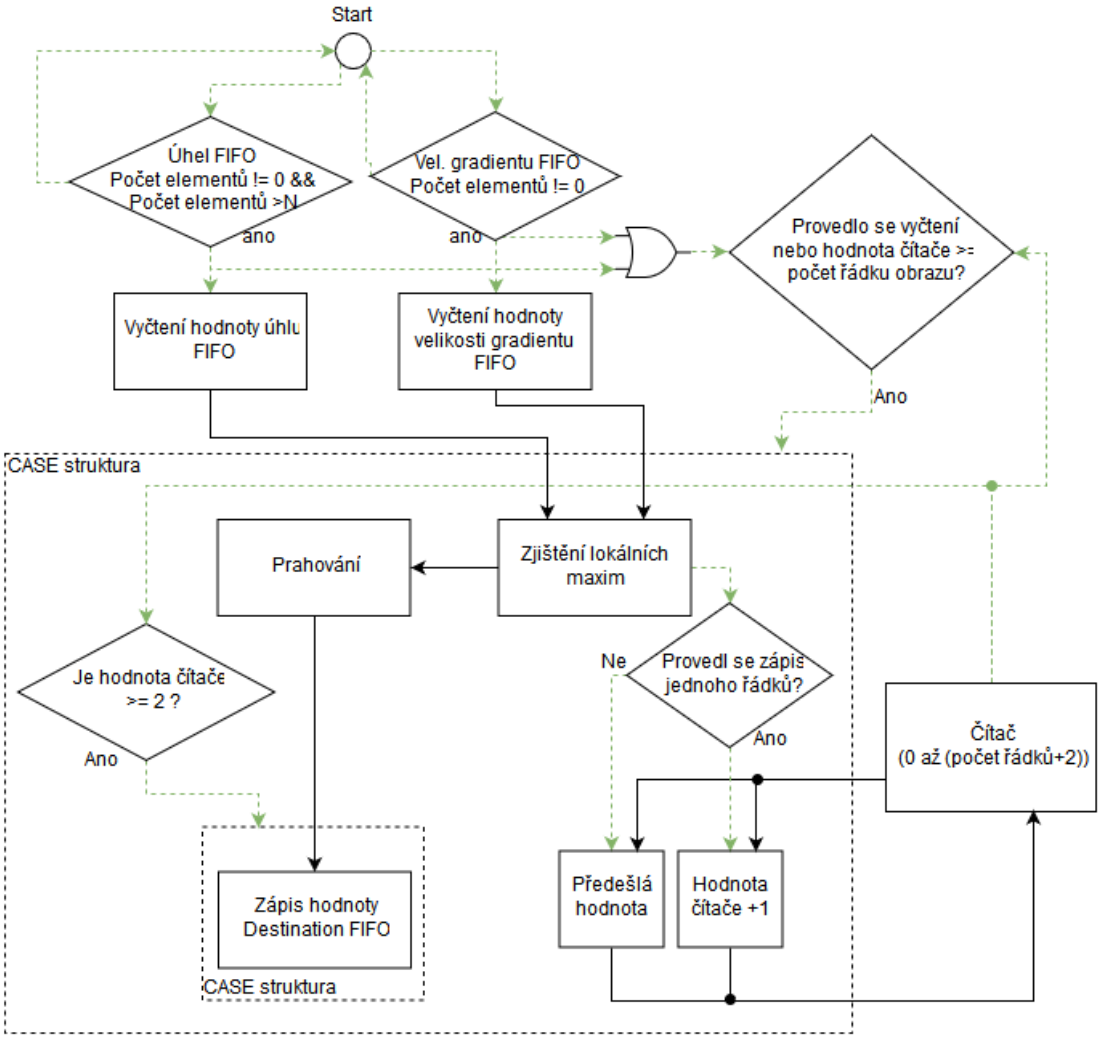
Čítač také slouží pro povolení hlavní Case struktury v okamžiku, když se dokončí vyčítání ze Source FIFO paměti, ale hodnoty pixelů jsou ještě uložené v pamětích Gaussova filtru a Sobelova operátoru. Celkem se provede zpracování tří řádků z pamětí, než se ukončí zpracování, a to způsobem zakázání přes Case strukturu, protože na FPGA zvolená smyčka je nastavena na neustálý běh. Case struktura se znovu povolí, až přijdou nová data.



Obrázek 17: Blokové schéma smyčky 1

Ve druhé smyčce (Obrázek 18) se vyčítá z FIFO paměti hodnota velikosti gradientu, v tomto případě FIFO hlavně slouží jako vyrovnávací paměť mezi dvěma smyčkami. U druhé paměti se vyčte hodnota úhlu gradientu, pokud daná FIFO paměť obsahuje alespoň N hodnot (jeden řádek obrazu). Potom dochází k povolení hlavní Case struktury, pokud byla vyčtena hodnota alespoň z jedné paměti, nebo počet vyčtených řádků je větší, nebo roven počtu řádků v obraze.

V Case struktuře se provede subVI pro hledání lokálních maxim a následuje prahování s hysterezí. Výsledná hodnota se zapíše do výstupní Destination FIFO paměti. Obdobně jako u první smyčky se zápis do FIFO paměti provede až se načtou dvě paměti ve funkcích, protože oba kódy operují s okolím 3x3. Opět se také provádí přičtení jedničky v čítači řádku, pokud se zpracuje jeden řádek. Hodnota čítač se používá úplně stejně jako v předešlé smyčce.



Obrázek 18: Blokové schéma smyčky 2

6.1.3 Implemetace převodu z RGB do modelu ve stupních šedi

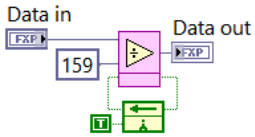
Jak vychází z předešlé podkapitoly, tak z DMA FIFO paměti se vyčte hodnota pixelu, která je datového typu U32. Jednotlivé barevné složky jsou zakódované v 8 bitech. Přičemž poslední složka A z RGBA se nepoužívá pro výpočet a je v tomto případě pro doplnění do velikosti datového typu.

Tato hodnota je potřeba rozdělit pomocí funkce Split number, která provádí rozdělení na bitové úrovní. Jednotlivé složky se potom převedou na datový typ FXP (s pevnou desetinou čárkou) a vynásobí se konstantami, a nakonec se sečtou pro získání výsledné hodnoty pixelu ve stupních šedi. Výsledná hodnota má 256 úrovní šedi a je vyjádřena 8 bity. Hodnota se zachovává v datovém typu FXP pro další zpracování.

6.1.4 Implementace konvoluční masky

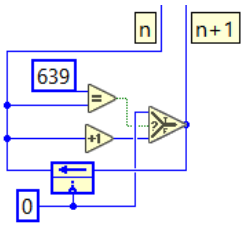
Kód pro FPGA je implementován na základě návrhu, který je popsán v předešlé kapitole 5. V této části jsou popsány klíčové části kódu a není tedy kompletně popsaný princip, který je celý popsaný v návrhu konvoluční masky. V tomto popisu implementace je tedy zahrnuta funkce Gaussova filtru a Sobelova operátoru, protože se v obou případech jedná o 2D konvoluci.

Ve smyčce SCTL nelze využít klasických funkcí pro dělení, odmocninu apod., je potřeba využít speciálních funkcí Hight Throughput. Tyto funkce je nutné správně nastavit pro správnou funkci kódu. V tomto případě je nutné nastavit, aby každou iteraci smyčky vyčetla jedna hodnota na výstup. Tímto nastavením, ale vznikne zpoždění. Velikost zpoždění se odvíjí od velikostí datových typů na vstupu funkce. Například na Obrázku 19, je funkce pro dělení se zpožděním 9 cyklů. Tyto funkce jsou využité u Gaussova filtru pro vydělení finální hodnoty pixelu, dále u Sobelova operátoru pro výpočet Arkus tangens úhlu gradientu a výpočtu odmocni u výpočtu velikosti gradientu.



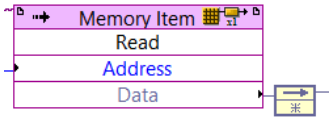
Obrázek 19: High Throughput funkce pro dělení

Jak je popsáno v návrhu, tak jsou využity paměťové bloky pro ukládaní řádků. Na Obrázku 20 je kód použitý pro indexaci paměťových bloků pro čtení a zápis. Na obrázku je indexace od 0 po 639 a jakmile dosáhne 639, tak se vynuluje. Tímto postupem indexace se vytvoří kruhový buffer.



Obrázek 20: Indexace paměti

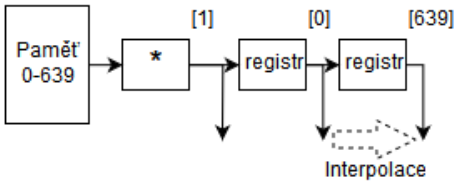
Na Obrázku 21 je ukázka paměťového bloku typu BRAM přistupující k definované paměti, v tomto případě provádí čtení z paměti. Na výstupu je přidaný posuvný registr, protože metoda čtení z paměti má zpoždění jedenu iteraci.



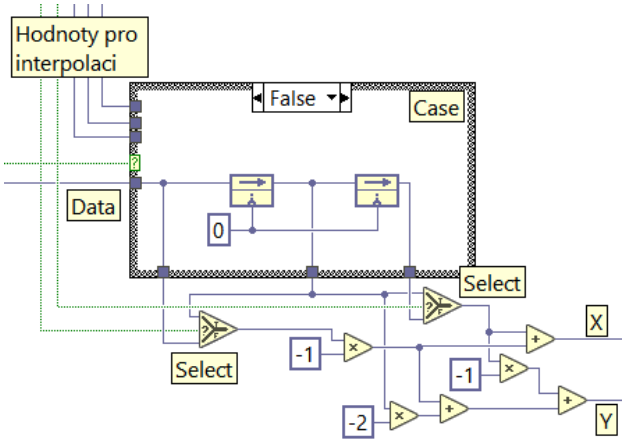
Obrázek 21: Paměťový blok pro čtení

Na Obrázku 23 je ukázka výpočtu třetího řádku konvoluční matice pro směr X a Y Sobelova operátoru. Case struktura slouží pro přenos hodnot z prostředního řádku kvůli interpolaci okrajových hodnot, volba se provádí pro horní a spodní řádek na základě hodnoty čítače řádků, který je popsaný v podkapitole 6.2.1.

Boční interpolace se provádí přes funkce Select, které na základě podmínky volí buď hodnotu z příslušných registrů, nebo berou hodnotu z prostřední hodnoty. Volba pro interpolaci bočních pixelů vzniká na základě hodnoty z čítače pro indexaci paměti (Obrázek 20). Pro přiblížení je na Obrázku 22 příklad interpolace, kde velikost paměti je 0-639 hodnot. Ve fázi kdy už jsou vyčtené dvě hodnoty z pozice 0 a 1 z nového řádku, tak za posledním registrem je poslední hodnota z předešlého řádku, zde se musí interpolovat hodnota z pozice 0 za poslední stupeň registru pro správný výpočet. Proto se musí povolit funkce Select, když je index pro čtení roven 2 a tím se provede interpolace pravé boční hodnoty a pro interpolaci levé hodnoty, když je index roven 1. K těmto konstantám 1 a 2 se musí přičíst ještě hodnota celkového zpoždění, která vznikne použitím funkcí High Throughput.



Obrázek 22: Interpolace bočních pixelů



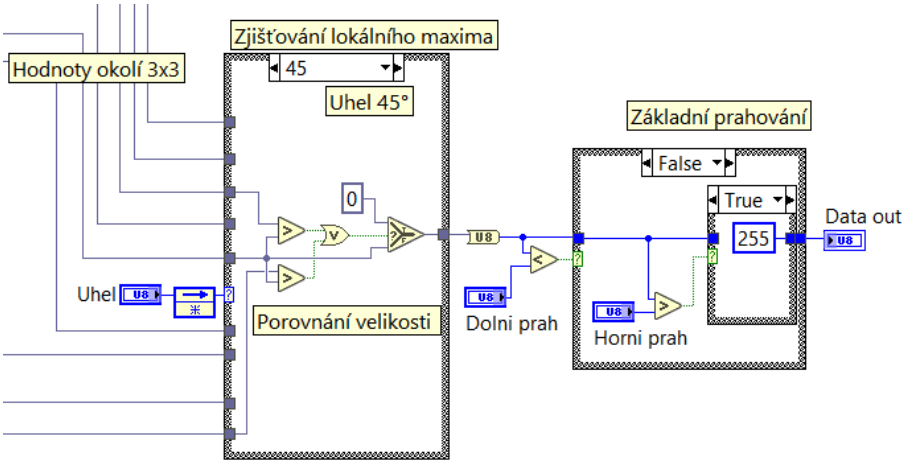
Obrázek 23: Implementace jednoho řádku Sobelova operátoru

Interpolace je potřeba provádět pouze v případě použití konvoluční masky, to znamená pro Gaussův filtr a Sobelův operátor. U hledání lokálních maxim a prahování s hysterezí není potřeba interpolace, protože se zde neprovádí výpočty, ale jenom se provádí rozhodování o úrovni pixelů.

6.1.5 Implementace hledání lokálních maxim

Do této funkce přichází data, která se vyčítají ze dvou FIFO pamětí jedna pro velikost gradientu a druhá pro úhel gradientu. Přičemž hodnota velikosti gradientu se ukládá do pamětí pro vytvoření okolí 3x3. A hodnota pro úhel gradientu se začne vyčítat z FIFO paměti, jakmile se načte jeden řádek hodnot velikosti gradientu a je tedy možné začít s porovnáváním hodnot.

Ukázka kódu je na Obrázku 24. Hodnota úhlu provádí volbu správného okna Case struktury pro zvolení správných hodnot, které se porovnávají se středovým pixelem. Z toho vyplývá, že není potřeba ukládat hodnotu úhlu, protože je nutný pouze pro středový pixel.



Obrázek 24:Kód pro hledání lokálních maxim a prahování

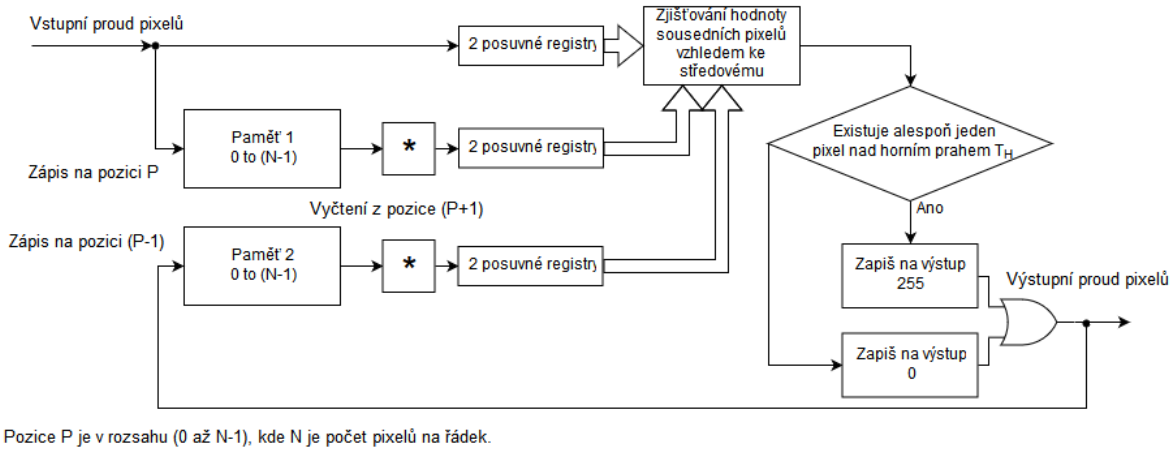
Z Case struktury vychází hodnoty pixelů, které jsou buď nastavené jako pozadí, nebo nezměněné. Na těchto hodnotách se pak provádí základní prahování, kde se opět přes Case strukturu nastavuje pixel buď na pozadí, když je pod dolním prahem, nebo jako hrana, když je nad horním prahem. Pokud je mezi prahy, tak se vypisuje na výstup nezměněn.

6.1.6 Implementace prahování s hysterezí

V této části je popis implementace prahování na FPGA. Princip implementovaného programu je znázorněný na Obrázku 25. U této funkce se předpokládá, že na vstupních datech je provedené základní prahování, které se vykonává hned za hledáním lokálních maxim a je tedy popsané v předešlé podkapitole.

V této části programu se provede uznání pixelů za hranu, nebo za pozadí. Podobně jako u Sobelova operátoru se využívá dvou pamětí a dvou posuvných registrů pro získání okolí 3x3. Získané hodnoty z okolí se porovnávají, jestli jsou nad horní prahovou hodnotou, to znamená, že jsou hranové. Pokud existuje alespoň jedna hodnota nad horním prahem, tak je středový pixel uznaný jako hranový. Hrana je definována nejvyšší úrovní 255 a pozadí jako 0. Pokud pixel není mezi horním prahem a dolním a byl už tedy v předešlém kroku uznaný jako pozadí, nebo hrana, tak se neprovádí zjišťování hodnot okolních pixelů, ale automaticky se hodnota posílá na výstup bez dalšího zpracování.

Výsledné pixely je potřeba ukládat do Paměti 2, protože už uznané hranové pixely je nutné porovnávat s dalším řádkem pro zachování kontinuity hran. Při ukládání do Paměti 2 se ukládá na pozici předešlou, než je to u Paměti 1. Takto nastavená pozice zápisu je z důvodu jednoho posuvného registru a takto nastavenou pozicí se kompenzuje vzniklé zpoždění. Vyčtení je stejné jako u předešlých algoritmů, to znamená z pozice o jednu vyšší, než je zápis.

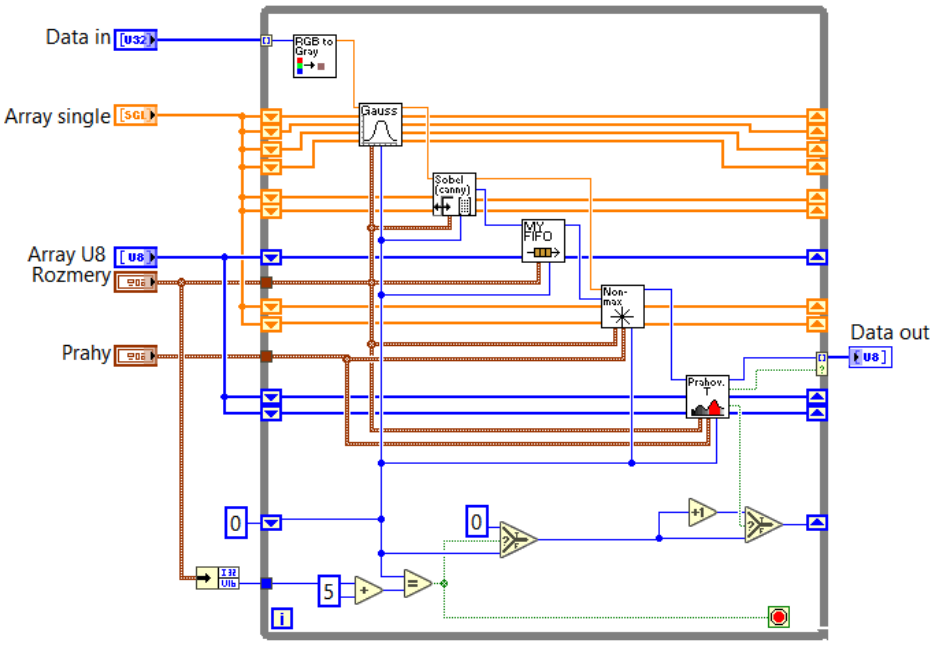


Obrázek 25: Blokové schéma prahování s hysterezí

6.2 Testy na PC

Testy, které se provádí na straně PC vycházejí z návrhu a implementace na FPGA. Přesto je několik rozdílů, které je potřeba provést z důvodů rozdílnosti platforem.

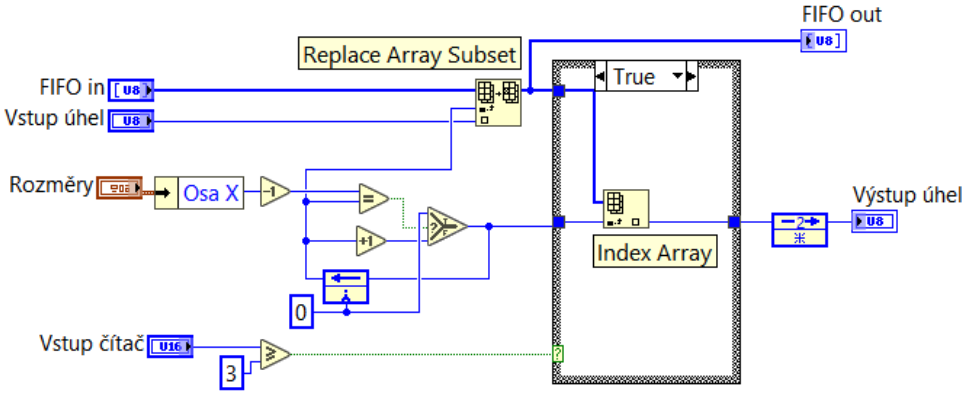
Program (Obrázek 26) není rozdělen do dvou smyček, jak je tomu na FPGA, ale je pouze v jedné smyčce. Data přichází do smyčky jako 1D pole s automatickým indexováním každou iteraci. Pro implementaci samotné konvoluční masky je použitý stejný postup jako je na FPGA, s tím rozdílem, že LabVIEW nemá paměťové bloky. Jako náhrada je využito pole, které se na začátku inicializuje s velikostí danou počtu pixelů na řádek na vstupu do smyčky. Do tohoto pole se potom ukládají jednotlivé hodnoty a v posuvném registru se uchová pole do další iterace smyčky.



Obrázek 26: Ukázka hlavní implementace Canny detektoru na PC

V případě FPGA se mezi smyčkami využívá dvou FIFO pamětí, jedna je pro velikost gradientu, kde se hodnota hned vyčítá a slouží jenom jako vyrovnávací paměť. A proto tato FIFO paměť není potřeba, protože kód je jenom v jedné smyčce v případě implementace na PC. Na druhou stranu druhá FIFO paměť, která je pro úhel gradientu se začne vyčítat až je zaplněna počtem pixelů na řádek. Tato FIFO paměť je potřeba implementovat ve smyčce na PC. Na Obrázku 27 je ukázka implementace FIFO paměti. Využívá se pole, které se indexuje od nuly po počet pixelů na řádek. Hodnoty FIFO paměti se mezi iteracemi se uchovávají v posuvném registru. Vyčítání z FIFO paměti začíná až čítač řádků bude nabývat hodnoty tři a více. Posuvný registr na výstupu z Case struktury slouží pro vykompenzování zpoždění, které způsobí dva

posuvné registry obsažené v následující funkci pro Hledání lokálních maxim. Kdyby tam nebyly, tak by se úhel přiřadil ke špatným velikostem gradientů a tím se znemožnila správná detekce hran.



Obrázek 27: FIFO paměť

Pro výpočty je využit datový typ Single s velikostí 32 bitů. Tento datový typ je zvolen záměrně, protože u většího datového typu Double by nebyla plně využita jeho přesnost, naopak u celočíselného typu by se výrazně negativně ovlivnila kvalita detekce hran. Obdobně jako na FPGA je využit čítač řádků, který je nejenom pro interpolaci okrajových pixelů, ale také pro ukončení smyčky.

Na rozdíl od FPGA, tak program na PC je možné rozdělit na více procesorových jader. V případě použitého počítače s procesorem se dvěma jádry, je stejný program spuštěný na obou jádrech s tím, že zpracovávají poloviční množství pixelů. Proto je potřeba 1D pole rozdělit na polovinu přes funkci Array Subset a jakmile se provede výpočet v obou funkcích, tak se znovu pole sloučí funkcí Apenned Array.

7 Vyhodnocení testů

Výsledné časy porovnávající efektivnost hardwarových platforem jsou v Tabulce 4. Kde čas za přenos zahrnuje v případě myRIO a IC dobu na zápis dat na FPGA, zpracování a nakonec vyčtení. V případě PC je tento čas jenom za provedení algoritmu. Čas celkem zahrnuje čas přenosu plus doba převodu snímku na 2D pole, na 1D pole a zpět. Na PC jsou testy provedené na jednom (single), nebo dvou procesorových jádrech (multi) a zpracování pomocí knihovní funkce pro Cannyho hranový detektor (NI).

Tabulka 4 Časy za zpracování Cannyho hranového detektoru

Platforma		Čas za přenos (ms)			Čas celkem (ms)		
		320x240	640x480	1280x720	320x240	640x480	1280x720
myRIO		2,8	13,0	40,6	7,0	37,4	123,0
IC		1,0	3,8	11,9	1,2	4,9	16,6
PC	single	26,6	105,5	310,3	26,8	106,5	313,4
	multi	20,6	79,4	280,5	21,0	80,3	284,0
	NI	17,2	68,6	198,0	17,7	70,0	202,1

Samotné provedení algoritmu na FPGA bez času na zápis a výpis dat na straně RT procesoru, lze vypočítat přibližně podle vzorce 7.1. Potom výsledné časy pro rozlišení 1280x720 jsou v Tabulce 5. Taktovací frekvence pro SCTL je na FPGA v případě IC je 90MHz a platformy myRIO je 55MHz.

$$t = perioda\ smyčky \cdot počet\ iterací \qquad (7.1)$$

Tabulka 5 Vypočítané časy za algoritmus na FPGA

Platforma	Čas za algoritmus (ms)
	1280x720
myRIO	16,75
IC	10,24

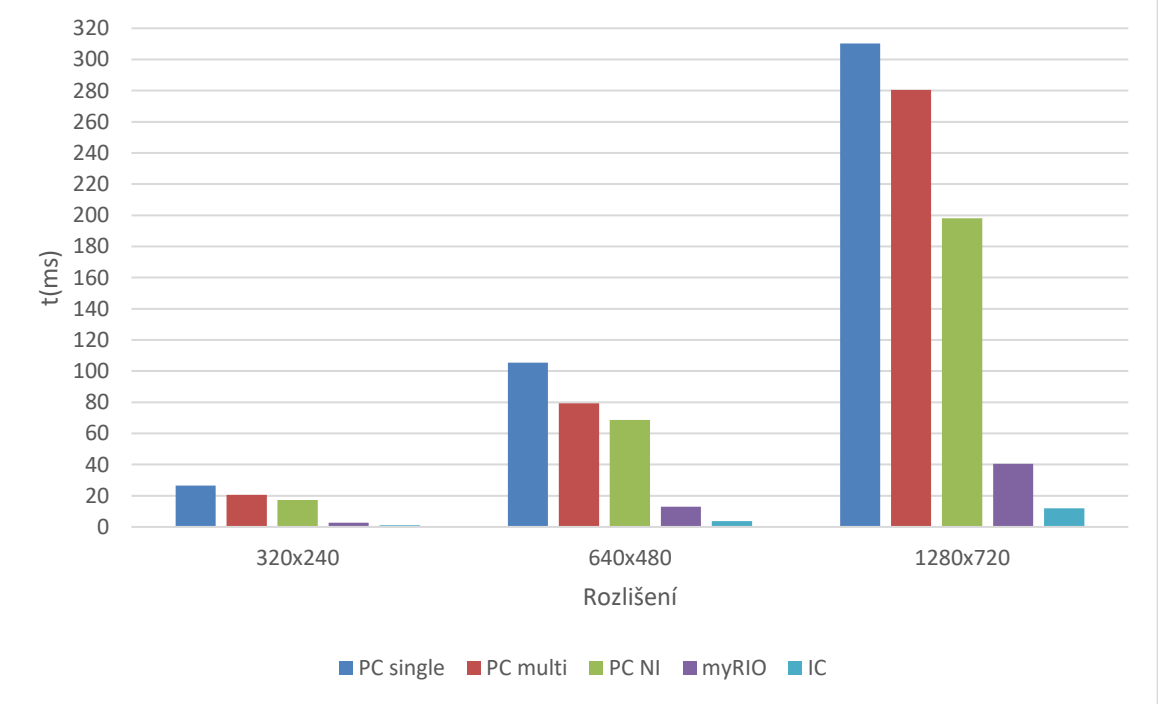
V Tabulce 6 jsou časy přepočítány na procenta, přičemž pro každé rozlišení je nejvyšší čas stanoven jako 100% a od něho se pak odvíjí kratší časy. Z tabulky jde vyčíst, že IC je napříč různým rozlišením ve stejném poměru k maximálnímu času. Velmi podobně vyrovnaný poměr má myRIO. V případě PC multi se u nejvyššího rozlišení zhoršil poměr k maximálnímu času. Z tabulky obecně vyplývá, že poměry časů k maximálnímu času jsou do určité míry stejné.

Tabulka 6 Časy za zpracování vyjádřené v procentech

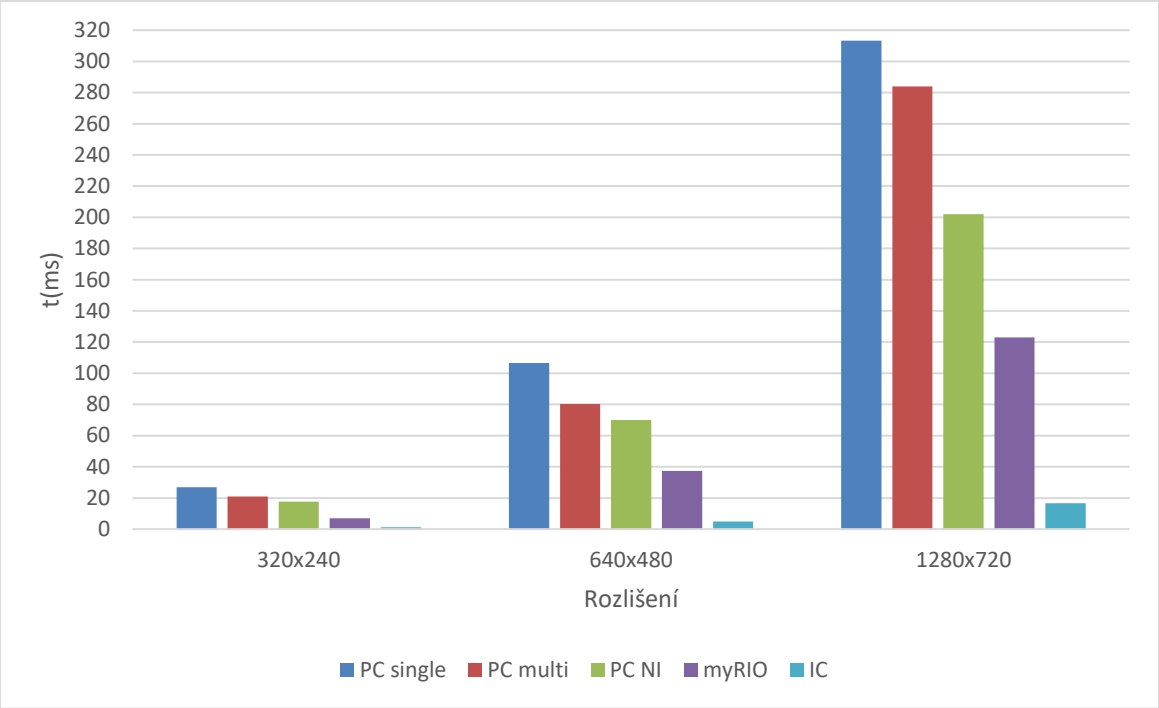
Platforma		Čas za přenosu v %			Čas celekem v %		
		320x240	640x480	1280x720	320x240	640x480	1280x720
PC	single	100,0	100,0	100,0	100,0	100,0	100,0
	multi	77,4	75,3	90,4	78,4	75,4	90,6
	NI	64,7	65,0	63,8	66,0	65,7	64,5
myRIO		10,5	12,3	13,1	26,1	35,1	39,2
IC		3,8	3,6	3,8	4,5	4,6	5,3

Porovnáním výsledných časů za zpracování algoritmu vychází výrazně rychleji na FPGA oproti stejnému algoritmu na PC. Graficky zobrazeno je pro čas za přenos v grafu na Obrázku 28. Výsledek testů nemusí rovnou implikovat, že FPGA bude vždy o tolik rychlejší, protože zvolený algoritmus nebude nejspíš ideální pro implementaci na PC. To ostatně ukazuje knihovní funkce od NI, která je výrazně rychlejší než vlastní navržená funkce.

Při porovnání změřených časů z Tabulky 4 a vypočtených časů z Tabulky 5, vychází v případě IC režie na straně procesoru na necelé 2ms, naproti tomu mnohem méně výkonnější myRIO má čas za režii ze strany procesoru necelých 24ms. Z toho vyplývá, že v případě myRIO je čas za zpracování nejvíce ovlivněn výkonem použitého RT procesoru a ostatních hardwarových komponent pracující pod procesorem. Dalším omezením může být způsob přenosu, kde u IC se mezi procesorem a FPGA používá PCI Express a u myRIO se používá přenos přes AXI protokol.



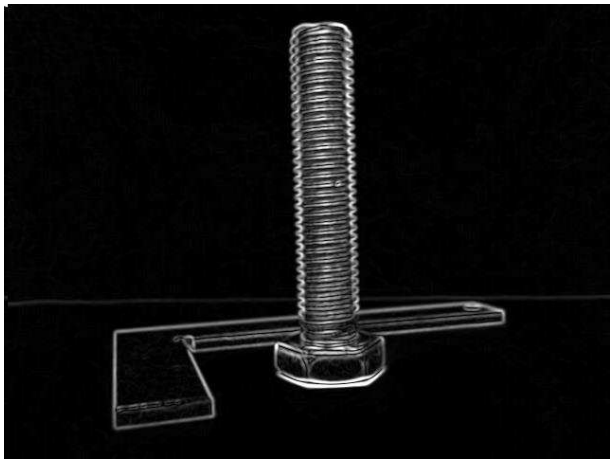
Obrázek 28: Graf času za přenos



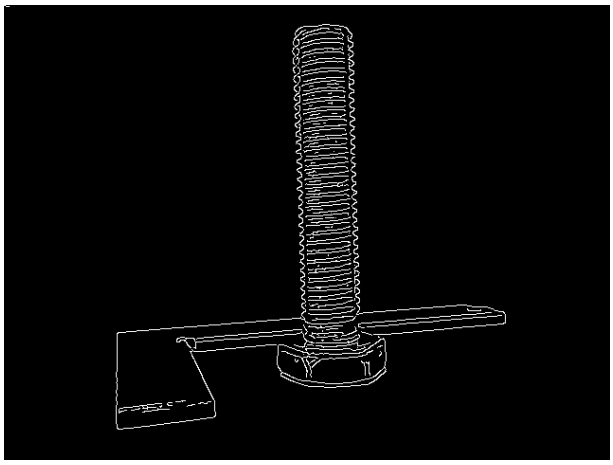
Obrázek 29: Graf času celkem

Na obrázcích 30 je ukázka snímku na kterém se provádělo zpracování. A na obrázcích 31 a 32 je ukázka zpracovaných snímků, kde je nejdřív ukázka výsledku ze Sobelova operátoru a potom výsledné zpracování Cannyho hranovým detektorem. Výsledky zpracovaných snímků vycházejí stejně při použití programu na PC i na FPGA.

Obrázek 30: Snímek před zpracováním



Obrázek 31: Snímek ze detekce hran Sobelem



Obrázek 32: Snímek z Cannyho hranového detektoru

8 Závěr

Cílem práce je porovnat celkem tři různé hardwarové platformy. První dvě jsou platformy myRIO a Industrial Controler, které využívají FPGA čip pro rychlé zpracování dat spolu s procesorem. Každá z platforem má rozdílné výkonové parametry. Třetí platformou je počítač. Jde tedy o porovnání klasického procesoru s FPGA čipem, přičemž v případě FPGA čipu se musí data dostat z paměti RAM, ke které má přístup procesor. Pro tento přenos je primárně zvolená metoda DMA FIFO, které využívá principu přímého přístupu do paměti pro přenos velkého objemu dat, aniž se musí data přenášet přes procesor. Procesor zajistí pouze spuštění této metody.

Jedny z hlavních rozdílů platformy myRIO a IC je přenos mezi procesorem a FPGA. V případě myRIO se přenos realizuje přes AXI protokol, které využívá propojení AMBA. A Industrial Controller využívá PCI Express. Rozdíly jsou i ve velikosti FPGA čipu.

Samotné programování se provádí v LabVIEW, respektive se využívá LabVIEW FPGA modulu. Tento modul umožňuje grafické programování HLS, které zavádí výrazné zjednodušení oproti využití nízko úrovňového programovacího jazyka HDL. Protože pro využití HDL je potřeba velmi dobrá znalost struktury a funkce jednotlivých částí FPGA. LabVIEW FPGA modul obsahuje rozsáhlý seznam knihovních funkcí pro zpracování obrazu, ale tyto funkce nelze použít, jelikož jsou tyto funkce uzamčené. Protože pro porovnání se musí napříč všemi použitými platformami využít stejný algoritmus. Proto je potřeba navrhnout celý algoritmus pouze pomocí základních prvků, aby šel realizovat jak na FPGA, tak i na procesoru.

Porovnání je prováděné na základě zvoleného algoritmu Cannyho hranového detektoru. Tento algoritmus je zvolen hlavně z důvodu, že algoritmus obsahuje několik způsobů zpracování obrazu. Jako je převod barevného modelu, konvoluční maska, prahování s hysterezí a jiné. Celkový algoritmus je potom poměrně náročný na zpracování, a to může vést k projevu nedostatků jednotlivých platforem.

Výsledkem jsou časy za zpracování. Kde jednoznačně nejrychlejší je program běžící na FPGA na Industrial Controleru. Druhým nejrychlejším je FPGA na platformě myRIO, kde ale výkon RT procesoru a ostatní části přidružené procesoru ovlivnily výrazně čas za zpracování. Rozdíl je také ve velikosti FPGA čipu, který umožnil taktovat smyčku na IC výrazně rychleji. Z výsledku se ukazuje, že samotný přenos dat na FPGA není místem zpomalení, ale hlavním zpomalením je výkon RT procesoru a velikost FPGA čipu. Zpracování na PC ukazuje, že rozdělení na více procesorových jader nemusí vždy dostatečně zrychlit zpracování, protože při zpracovávaní takto velkých polí je velká výpočetní režie na rozdělení a sloučení polí. Při porovnání knihovní funkce se ukazuje, že rychlost za zpracování výrazně závisí na volbě algoritmu. Proto při volbě vhodnějšího algoritmu jednotlivých částí Cannyho detektoru, by mohl dosahovat rychlejšího zpracování. Ale s největší pravděpodobností by nešel realizovat na FPGA pro porovnání, protože FPGA má větší omezí oproti LabVIEW na PC.

Jednou z výhod využití FPGA může být využití procesoru na jiné zpracování dat, když se mezitím zpracovává obraz na FPGA. Naproti tomu FPGA není příliš vhodné pro složité

aplikace, protože se tím výrazně zvyšuje náročnost na vývoj. Proto nejsou moc vhodné různé inteligentní algoritmy. A při programování aplikace na FPGA je zjištěno, že je velmi výhodné optimalizovat kód pro SCTL smyčku, která zajistí velmi rychlé zpracování.

Jedním z možných rozšíření může být použití grafické karty pro zpracování obrazu. Ta umožnuje podobně jako FPGA paralelní zpracování, ale oproti FPGA využívá velké množství procesorů, mezi které se rozdělí zpracování a tím se docílí paralelismu.

Použitá literatura

[1] NATIONAL INSTRUMENTS. NI FlexRIO FPGA Module Installation Guide and Specifications [online]. National Instruments Corporation, 2011 [cit. 2018-01-10]. Dostupné z: http://www.ni.com/pdf/manuals/373047b.pdf

[2] Camera Link Adapter Module for FlexRIO. In: National Instruments [online]. National Instruments Corporation, 2018 [cit. 2018-01-10]. Dostupné z: www.ni.com/cs-cz/shop/select/camera-link-adapter-module-for-flexrio

[3] PXI FPGA Module for FlexRIO. In: National Instruments [online]. National Instruments Corporation, 2018 [cit. 2018-01-10]. Dostupné z: http://www.ni.com/cs-cz/shop/select/pxi-fpga-module-for-flexrio

[4] Transferring Data between the FPGA and the Host (FPGA Module). National Instruments [online]. National Instruments Corporation, 2010 [cit. 2018-01-16]. Dostupné z: http://zone.ni.com/reference/en-XX/help /371599F-01/lvfpgaconcepts/pfi_data_transfer/

[5] Using Single-Cycle Timed Loops to Optimize FPGA VIs (FPGA Module). National Instruments [online]. National Instruments Corporation, 2011 [cit. 2018-01-16]. Dostupné z: http://zone.ni.com/reference/enXX/help/371599G01/lvfpgaconcepts/using_sctl_optimize_fpga/

[6] DOBEŠ, Michal. Zpracování obrazu a algoritmy v C#. Praha: BEN - technická literatura, 2008. ISBN 978-80-7300-233-6.

[7] HOTAŘ, Vlastimil. Úvod do problematiky strojového vidění. Část 2, Základy zpracování obrazu. Liberec: Technická univerzita v Liberci, 2015. ISBN 978-80-7494-202-0.

[8] SOJKA, Eduard. Digitální zpracování a analýza obrazů [online]. Ostrava: VŠB-Technická univerzita, 2000 [cit. 2018-04-21]. ISBN 80-707-8746-5. Dostupné z: http://mrl.cs.vsb.cz//people/sojka/dzo/digitalni_zpracovani_obrazu.pdf

[9] HLAVÁČ, Václav a Miloš SEDLÁČEK. Zpracování signálů a obrazů. Praha: Vydavatelství ČVUT, 2000. ISBN 80-010-2114-9.

[10] VLACH, Jaroslav, Josef HAVLÍČEK a Martin VLACH. Začínáme s LabVIEW. Praha: BEN – technická literatura, 2008. ISBN 978-80-7300-245-9.

[11] LabVIEW FPGA module [počítačový program], verze 2016, National Instrument

[12] NI myRIO-1900 User Guide and Specifications [online]. National Instrument, 2016 [cit. 2018-04-28]. Dostupné z: http://www.ni.com/pdf/manuals/376047c.pdf

[13] IC-317x User Manual [online]. National Instrument, 2017 [cit. 2018-04-28]. Dostupné z: http://www.ni.com/pdf/manuals/375285c.pdf

[14] Zynq-7000 All Programmable SoC Data Sheet [online]. Xillinx, 2017 [cit. 2018-04-28]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[15] 7 Series FPGAs Data Sheet [online]. Xillinx, 2018 [cit. 2018-04-28]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[16] FPGA Fundamentals. National Instrument [online]. National Instrument, 2018, 2012 [cit. 2018-04-28]. Dostupné z: http://www.ni.com/white-paper/6983/en/

Přílohy

Veškeré přílohy v podobě programů jsou na PC a myRIO jsou v přiloženém CD.